50325-0532 (Seq. No. 3861) *Patent*

UNITED STATES PATENT APPLICATION

FOR

DYNAMIC INFORMATION OBJECT CACHE APPROACH USEFUL IN A VOCABULARY RETRIEVAL SYSTEM

INVENTORS:

MICHAEL J. KIRKWOOD
SIMA YAZDANI
JERALD AL BASKAR

PREPARED BY:

HICKMAN, PALERMO, TRUONG & BECKER
1600 WILLOW STREET
SAN JOSE, CA 95125
(408) 414-1080

DYNAMIC INFORMATION OBJECT CACHE APPROACH USEFUL IN A VOCABULARY RETRIEVAL SYSTEM

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is related to and claims domestic priority from prior U.S. Provisional application Ser. No. 60/252,378, filed November 20, 2000, the entire disclosure of which is hereby incorporated by reference as if fully set forth herein. This application is related to prior non-provisional applications: Ser. No. not known, filed on March 30, 2001, Attorney Docket No. 50325-0527, entitled "Query Translation System for Retrieving Business Vocabulary Terms" by inventors M. Kirkwood et al.; Ser. No. not known, filed on March 30, 2001, Attorney Docket No. 50325-0528, entitled "Business Vocabulary Data Storage Using Multiple Inter-Related Hierarchies" by inventors M. Kirkwood et al.; Ser. No. not known, filed on July 18, 2001, Attorney Docket No. 50325-0529, entitled "Business Vocabulary Data Retrieval Using Alternative Forms" by inventors M. Kirkwood et al.; Ser. No. not known, filed on July 18, 2001, Attorney Docket No. 50325-0530, entitled "Techniques for Forming Electronic Documents Comprising Multiple Information Types" by inventors M. Kirkwood et al.; and Ser. No. not known, filed concurrently herewith, Attorney Docket No. 50325-0531, entitled "Multiple Layer Information Object Repository" by inventors M. Kirkwood et al., the entire disclosures of all of which are hereby incorporated by reference as if fully set forth herein.

FIELD OF INVENTION

The present invention generally relates to data processing in the field of electronic document creation. The invention relates more specifically to a dynamic information object cache approach useful in a vocabulary retrieval system or ontology-based system.

BACKGROUND OF THE INVENTION

Through economic growth, mergers and acquisitions, business enterprises are becoming ever larger. Further, large business enterprises in the field of high technology now offer ever larger numbers of products and services that derive from an increasingly large variety of technologies.

In this environment, managing the creation, use, and maintenance of the company's intellectual assets, such as products and technologies is an acute problem. As an enterprise grows, maintaining consistent usage of names of products and services throughout the enterprise becomes even more challenging. When an enterprise derives its business opportunities from research and development into new technologies or improvements of existing technologies, maintaining consistent usage of technology designations is a challenge, especially when there is disagreement or confusion about the uses, advantages or benefits of a particular technology. Such confusion can arise whether disagreements arise or not, as when there is no communication between different teams within an enterprise.

The World Wide Web is one communication medium that exacerbates the problem, by showing internal information to the enterprise's partners and customers. Large enterprises that own or operate complex Web sites or other network resources that contain product and technology information face a related problem. Specifically, ensuring consistent usage of product names and technology terms across a large, complicated Web site is problematic. A particular problem involves maintaining consistent use of terms when different parts or elements of the Web site applications are created or content is authored by different individuals or groups.

Based on the foregoing, there is a clear need for improved ways to manage one or more vocabularies of all company business practices and pertaining to all business terminology ("concept"), including but not limited to product names and technology terms.

In particular, there is a need for a way to structure stored information about those concepts so that it can be located and navigated easily regardless of who authored the information and where the information resides.

There is also a need for a system that can rapidly and efficiently select vocabulary concepts and related information from among a large volume of stored information that is inter-related by overlapping hierarchies, and deliver the selected information to another system for use in assembling electronic documents based on the selected information.

There is also a need for a way to deliver information distributed over one or more networks that is relevant to a user query based on the vocabulary information to individuals who are distributed among many groups of a large enterprise, or who are outside the enterprise.

There is also need for a system that is extensible or adaptable when new business practices, products or technologies are developed by diverse, distributed groups in a large business enterprise.

A system that meets the foregoing needs is useful for managing highly complex Web sites and similar information resources that can store, retrieve and deliver vast amounts of information to clients. Moreover, the information that is delivered can be provided in a personalized manner. Based on a client query, dynamic pages are constructed from individual information object components, and delivered in assembled form, with content responsive to the query. While such systems are powerful, they are also extremely complex and are required to rapidly deliver large amounts of information.

Accordingly, there is a need for a system having the foregoing characteristics that can provide improved performance. Specifically, a complex Web site based on the foregoing is required to perform as fast as possible to improve customer satisfaction and experience.

Past approaches involve caching Web site pages. An example of a commercial product series representing such past approaches is the Cisco 500 Series Cache Engine, e.g.,

the Cisco 550 Cache Engine. However, in these past approaches, electronic documents or Web site pages are created in advance in a fixed form ("static pages") and therefore are easily stored in a cache and delivered from the cache to clients. But when virtually all the pages delivered from the Web site are dynamic, and are created based on individual component information objects, use of traditional caching approaches is impossible because there is no way to cache a dynamically constructed page.

Another deficiency of such prior approaches is that they cannot cache multiple versions of electronic documents that are only slightly different, such as pre-rendered Web site pages that are "personalized" by association with a specific client.

Thus, there is a need for a way to cache dynamically generated electronic documents, such as Web site pages.

There is a particular need for a way to efficiently cache the information objects that form components of dynamically constructed electronic documents, in a manner that is compatible with the foregoing characteristics, so that requested information objects are rapidly and efficiently delivered from a data store that holds the information objects.

SUMMARY OF THE INVENTION

The foregoing needs, and other needs and objects that will become apparent from the following description, are achieved in the present invention, which comprises, in one aspect, a method of expressing knowledge electronically. One example of this aspect involves dynamically generating an electronic document, such as a Web document.

In one aspect, a method of dynamically generating an electronic document involves receiving a request to generate an electronic document containing information responsive to a user query based on one or more information objects that are organized in one or more hierarchical trees, wherein the query contains a concept and an information type that is part of a document, or document type. A cache of information objects is searched to identify one or more rows that match the query concept, the information type, and the document type. An intersection of the rows is determined, yielding a result set of rows. Matching information types are retrieved based on following index pointers in the rows of the result set, which point to an information object in an information object repository. Information objects logically represent any content, in any location. The electronic document is automatically created using the matching information objects and delivering the electronic document in response to the user query.

According to another aspect, a concept cache useful in a vocabulary management system stores references to individual information objects that can be retrieved and dynamically assembled into electronic documents. Information objects are organized in one or more hierarchical trees, and references to nodes in the trees are cached. A query processor receives a cache query from a delivery engine that is attempting to dynamically construct an electronic document with content that matches the query. Alternatively, one or more programmatic function calls defined in an application programming interface are called to process a query. The query contains a concept and an information type. The cache is searched

to identify one or more rows that match the query concept and the query information type. An intersection of the rows is determined, yielding a result set of rows. Index pointers in the rows of the result set lead to stored information objects, which are passed to the delivery engine. The delivery engine assembles the electronic document using the information objects. The information objects may represent any type of original object, e.g., Web services, HTML, images, applications, templates, etc.

Unlike past approaches that cache static pages, rapid delivery of dynamic pages is facilitated.

According to one feature, cache searches are also constrained by other factors, such as whether rows match a template type of an output document template, whether a particular class of user who issued the query is entitled to read the rows, language, etc.

In another aspect, the method includes managing a plurality of information chunks in one or more content management systems. Each chunk of the plurality of information chunks comprises a unit of data for storage and retrieval operations. A vocabulary database is also managed. The vocabulary database includes data structures describing atomic concepts among names in an enterprise-specific vocabulary, and a plurality of data structures describing relationships among the atomic concepts. The data structures describing atomic concepts include a first information object having data indicating a first reference to a first chunk in the content management system. The data structures describing relationships include a first relationship between the first information object and a second concept of the atomic concepts. All the foregoing are represented in a single cache.

In other aspects, the invention encompasses computer readable media, and systems configured to carry out the foregoing steps.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements and in which:

FIG. 1 is a block diagram that illustrates a hypothetical product type hierarchy according to one embodiment;

FIG. 2A is a block diagram that illustrates a networking solutions hierarchy including one or more concepts from the product type hierarchy of FIG. 1 according to one embodiment;

FIG. 2B is a block diagram that illustrates a non-binary relationship among concepts according to one embodiment;

FIG. 3 is a block diagram illustrating simultaneous multiple inter-related hierarchies involving a product type concept according to one embodiment;

FIG. 4A is a block diagram illustrating a vocabulary development server and external applications according to one embodiment;

FIG. 4B is a block diagram illustrating a creation layer of an information object repository and a resulting Web site according to one embodiment;

FIG. 4C is a diagram of a binary tree representation that can be modeled using one or more data structures stored in computer memory;

FIG. 4D is a diagram of a class hierarchy of an example object-oriented model;

FIG. 4E is a diagram of a data representation schema;

FIG. 4F is a block diagram of an example architecture of the VDS;

FIG. 4G is a diagram illustrating relationships among an access control list and nodes of a tree of the type shown in FIG. 4C;

FIG. 4H is a block diagram of a class hierarchy that may be used to implement an event mechanism, in one embodiment;

FIG. 5 is a block diagram that illustrates relationships involving a particular information object and other concepts in the vocabulary database;

FIG. 6A is a flow chart illustrating a method for managing an information object repository by generating and storing an information object according to one embodiment;

FIG. 6B is a flow chart illustrating a method for managing an information object repository by retrieving an information object according to one embodiment;

FIG. 6C is a flow chart illustrating a method for managing an information object repository by retrieving information content associated with an information object according to one embodiment;

FIG. 7 is a block diagram illustrating a management layer, a staging layer, and a Web server layer of an information object repository according to one embodiment;

FIG. 8A is a flow chart illustrating a method for generating a static Web page based on the Web server layer of the information object repository according to one embodiment;

FIG. 8B is a flow chart illustrating a method for generating a concept home Web page based on the Web server layer of the information object repository according to one embodiment;

FIG. 8C is a flow chart illustrating a method for generating a concept information type Web page based on the Web server layer of the information object repository according to one embodiment;

FIG. 8D is a flow chart illustrating a method for generating a concept document Web page based on the Web server layer of the information object repository according to one embodiment;

FIG. 8E is a flow chart illustrating a method for generating a concept search result Web page based on the Web server layer of the information object repository according to one embodiment;

FIG. 8F is a flow chart illustrating a method for generating an information chunk Web page based on the Web server layer of the information object repository according to one embodiment;

FIG. 9A is a flow chart illustrating a method for generating and managing a management layer of the information object repository according to one embodiment;

FIG. 9B is a flow chart illustrating a method for generating and managing a staging layer of the information object repository according to one embodiment;

FIG. 9C is a flow chart illustrating a method for preparing a Web site in a staging layer of the information object repository according to one embodiment;

FIG. 10 is a block diagram that illustrates a computer system upon which an embodiment may be implemented;

FIG. 11A is a block diagram of an example embodiment of a cache system;

FIG. 11B is a block diagram of a second example embodiment of a cache system;

FIG. 12 is a block diagram that illustrates in more detail an internal arrangement of a cache server;

FIG. 13 is a diagram of an example schema of tables that may be used in an embodiment; and

FIG. 14 is a block diagram of a distributed computing model in which hierarchies of information objects are distributed among multiple machines.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

A method and apparatus for storing business vocabulary data using multiple inter-related hierarchies are described. In the following description, for the purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent, however, to one skilled in the art that the present invention may be practiced without these specific details. In other instances, well-known structures and devices are shown in block diagram form in order to avoid unnecessarily obscuring the present invention.

1.0 BUSINESS VOCABLARY DATA PROCESSING

Business vocabulary terms are used to name products, product lines, technologies, people, processes, development efforts and other business activities of an enterprise. Some of the vocabulary terms are used only internally and some are used for interaction with the public to establish brand name recognition or to support precise communication of customer interests and orders. Terms related in meaning or form are used to associate related business products and activities in the minds of the users of those terms. For example, a device sold by an enterprise might be named Perseus, after a hero of Greek mythology, and a software program for executing on that device might be named Pegasus, after the winged horse Perseus rode. Similarly, different models of the Perseus device might be called AlphaPerseus and BetaPerseus, to show they are part of the same product line, while different versions of each model may be numbered, such as BetaPerseus 2.0 and BetaPerseus 2.4.

The present invention is based in part on a recognition that the business terms of an enterprise constitute an important type of business data that should be included in the automated data processing that the enterprise performs. This vocabulary data about the

products, services and activities of a business is a form of metadata for the products, services and activities of the enterprise. Those terms can be used to categorize the products, services and activities and to retrieve other data about those products, services and activities. The data structures employed to store, retrieve and process this metadata should account for the associations in meaning and form and support rapid associative or inferential search and retrieval.

2.0 VOCABULARY DEVELOPMENT FRAMEWORK

According to the present invention, the various terms that constitute the business vocabulary of an enterprise are modeled as nodes in a hierarchy called the MetaData Framework (MDF) or the Vocabulary Development Framework (VDF). In this framework, any business term that is derived from another particular business term is positioned in the hierarchy at a node that branches from the node of that particular business term from which it is derived. When the hierarchy is embodied in stored data with appropriate data structures and software programs, it is extremely useful in naming products and associating products with product lines.

For example, FIG. 1 shows a hypothetical product type hierarchy for a hypothetical enterprise that manufactures and sells network devices. In this hierarchy, node 102 is a root node representing network device products sold by the enterprise. Node 102 has three child nodes, 112, 114, 116 that are connected by arrows 105. The parent/child relationship is denoted by an arrow pointing from parent to child in FIG. 1. A relationship statement can be obtained reading from arrow head to arrow tail by the words "is a child of" or read in the opposite direction by the words "is a parent of." Thus node 112 is a child of node 102. Node 102 is a parent of node 112. In the product type hierarchy of FIG. 1, arrow 105 represents the product type parent/child relationship.

Node 112 represents the devices named "Perseus." In this embodiment, the name of node 112 includes "Perseus." Nodes 114, 116 represent devices named "Hercules" and

"Jason," respectively. FIG. 1 shows that the Perseus device comes in three models, "AlphaPerseus," "BetaPerseus" and "GammaPerseus," represented by the three nodes 122, 124, 126, respectively. The BetaPerseus model has evolved over time through versions 1.0, 2.0 and 3.0, represented by nodes 132, 142, 154, respectively. The names of these nodes are "BetaPerseus 1.0," BetaPerseus 2.0," and "BetaPerseus 3.0," respectively. BetaPerseus 2.0 also experienced some evolutions called "BetaPerseus 2.4" and "SuperPerseus," which are represented by nodes 152, 162, respectively.

This hierarchy consists of binary relationships; that is, each relationship requires one parent and one child. The product type relationships of FIG. 1 are constrained by a rule that each child may have only one parent. There is no rule restricting the number of children a parent may have in this hierarchy.

Various applications use the information in the VDF implementation to perform different functions for the enterprise. In one application, the VDF relationships in the illustrated hierarchy are used to determine that the product named "SuperPerseus" is actually a version of the BetaPerseus model that is based on version 2.4. In another application, the VDF names are used to help provide names for products as new products are developed by automatically including the product type and model name and by preventing the re-use of an existing version number. Embodiments of this application enforce a rule that each name shall be unique. The enterprise uses the VDF with other embodiments of such an application to enforce other naming rules, such as requiring the model name shall be part of the device name. In this case the ambiguous name "SuperPerseus" is not allowed, and is discarded in favor of the automatic name, "BetaPerseus 2.5", or some allowed variation of that, which is stored as the name of node 162.

The vocabulary data framework (VDF) captures simultaneous multiple relationships among names, products, solutions, services, documentation and activities for an enterprise. In particular, the VDF allows other relationships to be established between nodes

simultaneously with the product type relationship. Furthermore, the VDF allows any of these new relationships to involve more than the two nodes of the binary parent-child relationship already described. For example, it allows a trinary relationship among a father node, a mother node, and a child node. In general, the VDF allows N-ary relationships among nodes, where N is any integer equal to or greater than one and specifies the number of participants in the relationship.

In the more general realm of the VDF, the enterprise is considered a data domain that includes many atomic concepts that may be related. Atomic concepts include any data item involved in the enterprise that is not subdivided into separately referenced storage units. These atomic concepts include the business vocabulary for the enterprise data that is the subject of the present invention. Concepts include product type names, as in the above example, but also comprise names of projects and departments and references to paragraphs, chapters, documents, images, multimedia files, database records, database queries, network resources, citations, and network addresses, among other things. The concepts and relationships are captured in conceptual graphs which are organized primarily by a partial-order relationship, commonly known as a type hierarchy. The concepts are nodes in the graph and the relationships are connections between two or more nodes. Both concepts and relationships have enumerated characteristics in some embodiments.

The graph of FIG. 1 is an example of a conceptual graph ordered by its product type hierarchy of binary (parent-child) relationships. Whereas this is one example based on a product type hierarchy, the VDF allows for simultaneous and inter-related multiple type hierarchies, as is explained in more detail in the following sections.

2.1 MULTIPLE HIERARCHIES

As seen above in FIG. 1, concepts are related in a graph depicting product types. All the concepts in this graph are associated with one category of information in the enterprise data. That category is device product types, and that hierarchy relates concepts for products

that are related in development history, structure or function. However, enterprise data may include other categories or relationships. In general, multiple categories encompass the enterprise data. For example, some of the enterprise data for an enterprise that manufactures and sells network devices are related to equipment solutions for common networking problems encountered by customers of the enterprise. Products of the enterprise that are unrelated by the hierarchy of FIG. 1 nevertheless may be useful to solve the same kind of customer problem. Thus, such products relate to the same solution. To reflect these relationships, enterprise data also are placed in a category called networking solutions in one embodiment, and are organized in a solutions hierarchy that exists concurrently with the product type hierarchy.

FIG. 2A depicts an example hierarchy of concepts in a networking solutions category. In this example, three solutions expressed by the concepts "single server local net," "wide area net (2 sites)" and "private wide area net (3 to 8 sites)" are stored in the data structures representing nodes 212, 214, 216, respectively. All three nodes are children of the root node 202 having name "networking solutions" for this category of concepts. In the solutions type hierarchy of FIG. 2A, arrow 205 represents a networking solutions parent/child relationship. All the relationships represented by arrows in FIG. 2A are of this type. This relationship type differs from the product type parent/child relationship represented by arrow 105 of FIG. 1. Both relationship types are parent/child binary relationships, but they relate concepts in different categories.

As shown in the example of FIG. 2A, the product GammaPerseus, at node 232, is part of the equipment solution for single server local networks of node 212. Both AlphaPerseus, at node 234 and Jason at node 235 are part of the equipment solution for wide area networks connecting two sites, at node 214. BetaPerseus 2.0, at node 236, and Hercules, at node 237, are part of the equipment solution for private wide area networks connecting three to eight sites represented by node 216. Nodes 242 and 244 represent software products Pegasus 3.3

and a graphical user interface (GUI) upgrade that are installed on the BetaPerseus 2.0 device in addition to the default software that comes with that device.

The concepts at nodes 202, 212, 214, 216 may be placed in a category called networking solutions. The concepts 232, 234, 235, 236, 237 have already been placed in a category called enterprise device products; but they may also be placed in the category networking solutions. The concepts at nodes 242, 244 may be placed in a category called software products and also in the networking solutions category. FIG. 2A demonstrates that hierarchies of concepts in categories of enterprise data may be defined in addition to the hierarchy of concepts in the product type category, and demonstrates that categories may overlap.

Alternatively, non-overlapping categories are used in other embodiments. In such an embodiment, the relationship represented by arrow 205 is expressed as a relationship of a sub-component to a component of a networking solution, in which the sub-component may be a different category than the component. Rules can be expressed for the relationship. One possible rule is: software can be a sub-component of hardware, but not the other way around. The relationship type enforces this rule by specifying the role for a category participant. Similarly, a product can be a sub-component of a networking solution category but not the other way around.

2.2 NON-BINARY RELATIONSHIPS

FIG. 2B depicts a conceptual graph of an example non-binary relationship. This ternary relationship (also called a 3-ary relationship or three participant relationship) is useful for capturing the expertise of a person in the use of a product in a technology area. In this example, this relationship is used to state whether the expertise of a technician in the use of a product device within a technology area is of a quality that can assume values of "unknown," "poor," "average," "good," or "excellent."

The characteristics of the relationship type describe the number of participants and their category or categories. In this example the relationship type includes characteristics that indicate there are three participants, one from the user category, one from the technology category and one from the product device category. In addition, the characteristics of this relationship include at least one relationship value for storing the quality of expertise (unknown, poor, average, good, excellent). More details on defining and storing concepts and relationships are given in a later section.

The conceptual graph of this relationship in FIG. 2B shows three nodes 282, 284, 286 representing the three concepts, e.g., product BetaPerseus 2.0, technology private wide area network, and technician Jane, respectively. The three nodes are connected by a three-way, non-directional link 290. The link 290 includes an attribute named "quality" that takes on a value such as "good," indicating that Jane's expertise is good for using BetaPerseus 2.0 in private, wide area networks.

2.3 DOCUMENTATION CATEGORY

Another category of concepts that is extremely useful to an enterprise, for both internal and external users, is documentation concepts, which encapsulate elements of electronic or tangible documents. Concepts within a documentation category include headings, sections, paragraphs, drawings, images, information type, and document type, among others. Information type concepts express the type of content in terms of what it says; for example, information type concepts include but are not limited to "Introduction," "Features & Benefits," "Product Photo," "External Article Section" etc. Documentation concepts may be organized in a document type hierarchy that facilitates automatically generating accurate, complete, up-to-date visual or printed documentation pertaining to a particular product or service. Document type hierarchies include, for example, "Data Sheet," "Product Home Page," "Press Release," "Operator's Manual," and "External Article." For example, a device, like the hypothetical Beta Perseus 2.0, can be linked by a relationship to a

document type hierarchy describing the device, such as a "Perseus 2.0 Operator's Manual." As another example, a device, like the Beta Perseus 2.0, can be linked by a relationship to a section concept in a document type hierarchy describing the networking solutions of which the device is a component, such as a "Small Business Networking Press Release." More examples of document categories of concepts are given in a later section.

2.4 MULTIPLE INTER-RELATED HIERARCHIES

As seen in the above examples, a single concept, such as the device product BetaPerseus 2.0 may appear in several separate hierarchies. According to one embodiment, information defining the concept is stored only once in the VDF and relationships are defined to all other nodes to which the concept is adjacent in all the hierarchies.

Hierarchies may be implemented using a variety of programming techniques and data storage. One advantage of this approach is that changes to the concept can be made in only one location in the VDF and all hierarchies immediately become up-to-date and reflect the changes. This is also achievable at the database layer by using normalized tables. Further, all information generated based upon the hierarchies, such as documentation or screen displays, automatically reflects the changes.

Another advantage is that applications that retrieve the data can navigate one of the hierarchies to a particular concept and then immediately find the other hierarchies in which that concept occupies a node. Thus, a customer who has purchased a particular device product for one networking solution can determine other solutions that use that same device. The customer follows the current solution to the product and then reviews the relationships with other networking solutions of interest to the customer that utilize the device. When a networking solution of interest is found using the device, the newly found solution can be navigated above and below the node representing the device concept in order to determine what software and other devices, if any, are components and sub-components of the new solution. Further, the customer can search by solution and identify multiple products that can

satisfy the solution. The customer can then inspect each of the products, obtain its documentation, and determine which product is best suited to the customer's particular needs. In some embodiments, such information is synchronized with the customer's online profile so that it is available for later reference and can be personalized.

FIG. 3 is an example of a conceptual graph for multiple inter-related hierarchies that are associated with the device product BetaPerseus 2.0, based on the individual hierarchies and relationships of FIG. 1, FIG. 2A and FIG. 2B. The branch of the device product type hierarchy of FIG. 1 that includes the BetaPerseus 2.0 device concept appears as nodes 302, 304, 306, 308, 390, 310 and 312 linked by the device product type, binary parent/child relationships 301. The branch of the device networking solutions hierarchy of FIG. 2A that includes the BetaPerseus 2.0 device appears as nodes 322, 324, 390, 332 and 334 linked by the networking solutions type, binary parent/child relationships 321. The 3-participant expertise relationship 391 links the node 390 for the BetaPerseus 2.0 to the concept "Jane" at node 346 and the concept "private wide area networks" at node 356. Also shown is that the concept "Jane" at node 346 is a child of the concept "technicians" at node 344 which is a child of the concept "users" at node 342. These nodes are linked by user type, binary parent/child relationships represented by arrows 341. Also shown is that the concept "private wide area networks" at node 356 is a child of the concept "wide area networks" at node 354 which is a child of the concept "technologies" at node 352. These nodes are linked by technology type, binary parent/child relationships represented by arrows 351.

The BetaPerseus 2.0 concept at node 390 is linked to the following nodes in multiple inter-related hierarchies. The BetaPerseus 2.0 concept at node 390 is a product type child of the BetaPerseus 1.0 concept at node 308, as represented by arrow 301d. The BetaPerseus 2.0 concept at node 390 is a product type parent of the BetaPerseus 2.4 concept at node 310, as represented by arrow 301e, and the BetaPerseus 3.0 concept at node 312, as represented by arrow 301f. The BetaPerseus 2.0 concept at node 390 is further a solutions type sub-

component of the private wide area net (3 to 8 sites) concept at node 324, as represented by arrow 321b. The BetaPerseus 2.0 concept at node 390 has solutions type sub-components of the Pegasus 3.3 software tools concept at node 332, as represented by arrow 321c, and the management software GUI upgrade concept at node 334, as represented by arrow 321d. The BetaPerseus 2.0 concept at node 390 has two companion expertise type participants as represented by link 391; one at Jane represented by node 346 and one at private wide area networks represented by node 356. In all, the example concept at node 390 has 6 binary relationships and one ternary relationship with eight nodes in four hierarchies (product type, equipment solutions, users and technologies). Each of the concepts and relationships may be represented using stored data in a database or appropriate programmatic data structures.

Many of the other nodes in FIG. 3 may have relationships with other hierarchies in addition to the relationships shown. These other relationships are omitted so that FIG. 3 and this discussion are more clear. Multiple relationships similar to the examples listed for node 390 may be defined for these other nodes.

2.5 ROOT CONCEPTS

At the top of each hierarchy for each category is a category root node representing the category root concept from which all the other concepts in the category branch. For convenience in navigating from one category to the next, each of the category root nodes is made a child of an enterprise data root node representing a top-level pseudo-concept for the enterprise data. In one embodiment, the pseudo-concept is "Vocabulary," and every node related to the Vocabulary concept by a direct "child of" relationship is a root node representing a root concept for one category.

2.6 IMPLEMENTATION OF THE VDF

FIG. 4A is a block diagram illustrating a vocabulary development server and external applications according to one embodiment.

According to one embodiment, the VDF is implemented in the form of one or more software elements that programmatically observe the following rules. The desired attributes of the VDS are derived from the Ontology model wherein the real world Objects are modeled as atomic concepts and relationships among the concepts.

1. A Concept is an atomic unit of a company's intellectual property, known as and represented by a Node.
2. A Concept is a normalized name, that is, one and only space character separates the words. Example: "Book title", "Author_name", "Book_written__by".
3. A Concept may have zero or more properties, known as attributes.
4. An Attribute is a (Name, Value) pair, where Name cannot be duplicate in a Node.
5. A set of concepts arranged in a hierarchy represents a taxonomy and may be represented logically or in memory as a Tree. The tree is composed with certain rules:
   a. A tree has a root node, called Category node.
   b. A Category is a concept and it has a name.
   c. A Category cannot be duplicated in a system
   d. A Category has a special node called Orphan Node. When a Concept node is deleted, the children Concept nodes are moved under the Orphan node. The Concepts under the Orphan Node are known as Orphans.
   e. The category node is directly attached to the pseudo node, Vocabulary Node.
   f. A tree should not have duplicate names/concepts. The requirement may be to allow case-sensitivity.

6. A node should have one and only one parent node, except the pseudo node that does not have a parent node.

7. A Concept in a hierarchy may inherit the properties from the parent node unless the property value is set in the node or the parent attribute(s) is not exposed to its children. The inheritance goes all the way up to the pseudo node, known as Vocabulary node.

8. A Concept may have relations with one or more other concepts. The relationships may exist across other taxonomy.

9. A relationship is a Concept whose name is assigned by the system, except for the root relationship nodes, known as Relation Types. A relationship is also referred as Relationship Instance or "Instance."

10. A relationship is a special node that is in addition to having the qualities of a Concept; it has two or more references to other concepts, known as Relationship Participants (in short Participants).

11. A Relationship participant has a Role name and a reference. The role name is simply an identifier for the reference. A Role name cannot be duplicate in a Relationship.

12. A relationship has one and only parent, Relation type.

13. A relation type is a root node for one or more relationship nodes and is a relationship node itself. So the participants for the relation type, known as type participants are the references to Category nodes and Relation type nodes. The relation type node is a template and dictates the possible participants for a relationship instance and the role names for the participant candidates. A relation type is another hierarchical taxonomy, mostly at single level.

The rules for creation and existence of a relationship are:

1. The relationship must have participants from the taxonomy of Categories or Relation types that the relation type specifies.
2. The role name of the instance participant should also match with the specification in the relation type.
3. In a relation type, there should not be any two instances having the same set of participants.
4. If a type participant is pseudo concept Vocabulary, then the relationship instances can have concepts from any taxonomy.
5. A relation type cannot be duplicated in a system.
6. When a Relation instance is removed, the node is simply removed from the Relation type.
7. When a Relation type is removed, all the relation instances and the type are removed from the system.

When a concept is removed, the following processing steps occur:

1. All the relationship instances that this concept is one of participants are removed.
2. The hierarchies of nodes below this concept node are moved to the Orphan node of the Category node.
3. The Concept node is removed from the system.

When a category is removed, the following processing steps occur:

1. All the relationship instances and Relationship types that this category is one of participants are removed.
2. All the Concept nodes including Orphans are removed from the system with relevant relationships.
3. The Category node is removed from the system.

In one embodiment, the VDS is configured in a way that offers good performance in terms of support for a large volume of simultaneous requests, extensibility and adaptability to new business requirements. The VDS provides security and internationalization support for concepts and relationships.

One embodiment uses a rule-base and declarative computation approach to express the concepts, relationships and rules of the VDF. This approach may be implemented using a high level computer programming language. In one embodiment, the approach is implemented using a logical processing language such as PROLOG™. The high level logical processing language translates statements declaring types and statements expressing rules about combining types into another language, such as the C programming language, that can be compiled and run on a large variety of general-purpose computer platforms. This approach relies on the inference power of a declarative engine and reduces coding and implementation that may impose a performance penalty.

In another approach, the taxonomy of hierarchical concepts and their relationships can be modeled as an in-memory tree data structure. FIG. 4C is a diagram of a binary tree representation that can be modeled using one or more data structures stored in computer memory. This model captures the business logic and is supplemented with constraints placed on the data model as programming logic. One example of such rule could be "a child concept should have one and only one parent." This approach is fast and efficient but has limitation that it uses up the main memory considerably. A file based or database based LRU (Least Recently Used) algorithm implementation would overcome this limitation.

Referring now to FIG. 4C, each of the top-level nodes 491 under the Vocabulary pseudo concept node 490 is a Category node, which implements the additional business logic and facilitates fast lookup and retrieval of concept nodes. Similarly, Relation type node 492 implements additional constraints on the relationship instances and facilitates fast responses to queries of n-ary relationships. A performance response of approximately less

than 1 millisecond is achieved by having appropriate indices in the Category and Relation Type nodes 491, 492. A simple Hash Map or a balanced tree data structure could model the in-memory index.

An example for retrieval could be as follows. Assume that the system receives a query getParticipants() with a set of arguments the identify participants in a set of relationships. The system is expected to return the matched relationship instances. One approach would be to go through each of the relationship instances and check for the match. When there are millions of instances, this would be slow. Accordingly, in a preferred approach, the following steps are followed to retrieve the information fast and efficient.

The system maintains an array of relationship instances for each of the participants on the system. The array of instances that have minimum index length is chosen given the query participants. Each element in the array is checked for a match by comparing the query participants and the relationship participants. This is quick and involves less computation. As an example, referring again to FIG. 4C, a fictitious hierarchical model is shown. The Relation1 is a relation type that has index for each of the participant for all the relation instances. Example indices but provided few and the names are given as numbers for explanation.

Participant	Instances
1	54001, 54002, 54011, 54202, 54301, 54042
4	54000, 54001, 54042
8	54001, 54202, 54900, 54301, 56899, 63629

Now consider a query getRelationship("Relation1", {"4", "8"}) wherein the API returns the relation instance names. A hash table lookup using the relation type name 'Relation1' would return the Relation type Object. The relation type object contains a hash table of participants and arrays of the relationship instances as in the table. A look up on this table using "4" and

"8" returns 2 arrays containing relation instances. Now the implementation chooses the array for "4" as it has minimum instances to compare. The system checks "54000" to determine if it is in the list of participant "8"; since it is not present, it is ignored. The system checks the value "54001" in the list of participant "8," and there is a match; and we exhaust all the elements. The result set is a list with one element "54001."

Embodiments also provide flexibility and adaptability to the new requirement by having an Object Oriented Data Model, which can be implemented in any Object Oriented Language like C++ or Java. FIG. 4D is a diagram of a class hierarchy of an example object-oriented model, in which a class "VDFNode" is the base class that models the tree data structure. Because all the other nodes are inherited from VDFNode, flexibility is provided. For example, causing a Relation Type to have another Relation Type as participant, could be done by having the type participant as VDFNode.

The core classes of an example implementation of the VDS in the Java language are described here. The implementation is shown in Java language, however the implementation could be done in any higher-level programming language, e.g., C, C++, etc.

A ConceptName represents the name of a Concept, as in this code example:

```
public interface ConceptName extends Name {
}

public interface Name {
        public String getName();
        public void setName(String name);
}
```

An Attribute class encapsulates a (Name, Value) pair, as in:

```
public interface Attribute {
        public Name getName();
        public void setName(Name n);
        public Object getValue();
        public void setValue(Object v);
}
```

A VDFNode class implements tree data structure and provides the basic tree operations. It also provides all the get, add and set APIs for attributes. However the set APIs pushes the action upwards in the tree hierarchy, which allows the calls to be trapped by the root level nodes, CategoryNode and RelationTypeNode for enforcing business logic and Access Control. An addChild method goes all the way up until it finds a Node that does checks and calls the actual add implementation _addChild() in VDFNode.

```
public abstract class VDFNode {
        private VDFNode parent=null;
        private VDFNode firstKid=null;
        private VDFNode sibling=null;

        private int nodeID;
        private Set attributes=null;

        public void changeAttribute(Attribute attr)  {
                // some implementation
        }

        public void addAttribute(Attribute attr)  {
                // some implementation
        }
        // more attribute related APIs

        public void addChild(VDFNode n) {
                setChild(this, n);
        }

        public void setChild(VDFNode parent, VDFNode child) {
                VDFNode p=getParent();
                if (p==null)
                        throw new Exception("business logic not found in the
hierarchy");
                p.setChild(this, child);
        }

        protected void _addChild(VDFNode n) {
                // imple..
        }

        protected void _removeChild(VDFNode child) {
                VDFNode prev=null;
                if (firstKid==child)
```

```
                    firstKid=child.sibling;
                else if ((prev=child.getPrevSibling())!=null)
                    prev.sibling=child.sibling;
                child.sibling=null;
                child.parent=null;
        }
        //more APIs

    }
```

A ConceptNode is a VDFNode and has a Normalized Name. Again a call to a set/add method pushes the call up to the root node which sets/adds the Object after the constraint checks.

```
    public class ConceptNode extends VDFNode {
        Concept concept=null;

        public void setConcept(Concept c) {
            setConcept(this, c);
        }
        public void setConcept(ConceptNode node, Concept c) {
            VDFNode p=getParent();
            if (p==null)
                throw new Exception("business logic not found in the
        hierarchy");
            if (p.getType()!=Constants.ConceptNode)
                throw new Exception("Invalid node in the hierarchy");
            p.setConcept(node, n);
        }

        protected void _setConcept(Concept c) {
        // this.concept=c;
        }
    }
```

A CategoryNode is a root node in taxonomy of concepts. It implements the business rules related to Concepts as stipulated in VDS rules. Here is an example: the setConcept() method is implemented here to check for duplicate Concept Name and to set the concept to the target ConceptNode, node. The root node implementation in CategoryNode and RelationTypeNode uses Read/Write Lock Object for efficiency that allows multiple reader

threads to go through, instead of Java synchronization that allows single reader thread to pass through the critical path.

```
public class CategoryNode extends ConceptNode implements Comparator {

        protected ReadWriteLock rwLock=new ReadWriteLock();
        protected boolean ignoreCase=true;
        protected TreeMap concepts=null;

        public void setConcept(ConceptNode node, Concept c) {
                rwLock.writeLock();
                try {
                if (!concepts.contains(c.getName())) {
                        concepts.put(c.getName(), node);
                        node._setConcept(c);
                }
                else
                throw new Exception("Concept duplicate. ");
                } finally {rwLock.releaseLock();}
        }
}
```

RelationParticipant has a reference to a VDFNode and role name for the reference.

```
public class RelationParticipant  {

        private VDFNode participant=null;
        private String roleName=null;
}
```

A RelationNode is the base class the relationship classes. It captures set Relation Participants with their Role Names. The field 'role_participants' is 2 dimensional array of (Role name, VDFNode). The class provides all the APIs for setting and getting the values in the collection.

```
public abstract class RelationNode extends ConceptNode {
        private HashMap role_participants=null;

        public void addParticipant(RelationParticipant part) {
                setParticipant (this, part);
        }
        public void setParticipant(RelationNode node, RelationParticipant part) {
                VDFNode p=getParent();
```

```
                if (p==null)
                        throw new Exception("business logic not found in the
        hierarchy");
                if (p.getType()!=Constants.RelationNode)
                        throw new Exception("Invalid node in the hierarchy");
                p. setParticipant (node, part);
        }

        protected void _setParticipant(RelationParticipant part) {
                // role_participants.put(part.getRole(), part) ;
        }

}

public class RelationTypeNode extends RelationNode {

        private ReadWriteLock rwLock=new ReadWriteLock();
        private HashMap relations=new HashMap();
        public void setParticipant (RelationNode node, RelationParticipant part) {
                rwLock.writeLock();
                try {
                List list=node.getParticipants();
                list.add(part);
                if (existsRelation(list))
                        throw new Exception("relation already exists with same
                        participants. ");
                node._setParticipant(part);
                VDFNode p=participant.getParticipant();

                // build cache in advance
                List plist=(List)relations.get(p);
                plist.add(part);
                } finally {rwLock.unlock();}
        }

}
```

2.6.1 DEFINING CONCEPTS

In one embodiment, a statement declaring that the phrase BetaPerseus 2.0 is a concept is presented in a high level logical processing language by the expression:

```
new Concept('BetaPerseus 2.0');
```

Similar expressions are used to enter the other concepts in the vocabulary.

The concept may have several attributes besides the phrase that defines it. For example the concept may have a creation date and an author. Attributes of a concept are presented with the following expression:

concept.addAttribute(new Attribute('creation' , '9/19/2000'));

2.6.2 DEFINING RELATIONSHIPS

The relationships that constitute a hierarchy connect one concept to one or more other concepts. Relationships are defined with the following expression:

new RelationTypeNode("prod_can_have_doc", 2);

where "prod_can_have_doc" is a relationship type and "2" is a value associated with the parameter type, i.e., in this example, a product can have 2 documents associated with it.

relationType.addChild(new RelationIntanceNode(new VDFNode[] {conceptNode1, conceptNod2}));

2.6.3 RETRIEVING RELATIONSHIPS

relationType.getRelationship("marketDoc", "BetaPerseus 2.0" ,

"http:///www.Enterprise.com/literature/ devices/catalog/Chap2/");

2.6.4 PERSISTENT DATA STORAGE

Changes in the VDS system need to be recorded on a permanent store for recovering and backup. VDS uses RDBMS for its persistent storage. FIG. 4E is a diagram of a data representation schema in the form of a fixed set of normalized tables that may be stored in persistent storage. The arrangement of FIG. 4E offers flexibility to model n-ary relationships and m by n level hierarchy. VDS system generates unique ID for each of the nodes as they are created in the system through adding a concept or relationship. These IDs are used as the primary keys in the database tables. The implementation commits the changes to the persistent store at specified interval as a batch update for enhancing performance. This must be accomplished at greater care to avoid loosing changes. This achieved by having a separate

Thread that maintains the changes so as to update them to the persistent store at regular interval. The changes are written to transaction.dat, which accumulates the events as they happen in the system, and transaction_history.dat, which maintains the history of transaction files that are to be merged, and that are already merged successfully to the database. The format of the transaction.dat is: command!argument[!argument]* as shown below:

```
1018!270560607! status_date!2001 06 14 16:56:56
1016!570560601!REL1!2
1017!57056060l!57067223
```

When the Thread wakes up to synchronize the database, it moves transaction.dat under a directory with time-stamp part of filename as in 2001/6/14/19_58_ transaction.dat. The thread runs through the lines of the files, composes SQL prepared statements and does the batch updates to the database. As one batch succeeds the lines involved in the batch update are prefixed with '+' sign to indicate that they are merged with the database. This way the server could merge the uncommitted changes to the database incase of error.

```
+1018!270560607!deploy_status_date!2001 06 14 16:56:56
+1016!570560601!Prod_PCR!2
+1017!57056060l!57067223
```

The transaction _history.dat is a quick index to the thread to find the files that are not fully committed. When the thread merges the changes, it marks appropriate entry in the history files with '+'. A typical history file looks like this:

```
+/opt/httpd/root/apps/mdf-sr/7213copy/2001/6/14/16_56_yacs_tran.dat
+/opt/httpd/root/apps/mdf-sr/7213copy/2001/6/14/17_57_yacs_tran.dat
 /opt/httpd/root/apps/mdf-sr/7213copy/2001/6/14/18_58_yacs_tran.dat
 /opt/httpd/root/apps/mdf-sr/7213copy/2001/6/14/19_58_yacs_tran.dat
```

The Vocabulary Development Server (VDS) is one or more processes that provide management of and access to the enterprise data in the vocabulary database to other

processes in an enterprise data processing system. Herein, the vocabulary database is also called the VDS Concept Database.

FIG. 4A is a block diagram illustrating a vocabulary development server and external applications according to one embodiment. FIG. 4F is a block diagram of an example architecture of the VDS according to another embodiment. Services provided by the VDS to clients and applications include vocabulary management and administration. Vocabulary related services are exposed to remote clients through a Metadata Access Protocol (MAP) over TCP/IP or RMI. Administration is a non-functional requirement but a desired to have features that allows remote monitoring, server fine-tuning.

MAP is designed for performance enhancement over RMI based approach. MAP is language neutral protocol wherein the request and response are transmitted over TCP/IP as tokens. The client application must know to assemble the tokens into the desired return result. The request format is:

Command_Identifier!Arguments_separated_by_!

Example

getChildConcepts!Category1

The response format is:

If Request succeeds , the format is:

+OK command_code

<responses in single or multiple lines>

<CRLF>

If Request is failed, the format is:

-ERR!error_code!error_message_in single line

2.6.5 SECURITY

VDS is a knowledge repository for storing and establishing Cisco's standard for concept categorization and their relationships. To provide a controlled access and

modification to the vocabulary, VDS implements two levels of security, Authentication and Authorization.

For Authentication, VDS supports simple username/password authentication mechanism and will service MAP over SSL in the future. It can be configured to use LDAP service to validate the user. The server also supports generic accounts (for which usernames do not exist in LDAP) through its internal authentication module.

For Authorization, VDS supports access control on all the nodes. Access Control List (ACL) is modeled within VDS as a set of categories and relation types. FIG. 4G is a diagram illustrating relationships among an access control list and nodes of a tree of the type shown in FIG. 4C. Permission on a node is granted to an action provided one the following is true:

1. If the access_mode on the node allows the action
2. If the user is in the group that has the required permission on the node.
3. If the permission on the parent of this node satisfies one of the above.

2.6.6 INTERNATIONALIZATION

The VDS system stores the names in double-byte character set. This achievable if the implementation language supports (like Java) or by taking care of it in the implementation by storing the name in appropriate data structure.

2.6.7 VDS EVENTS

Events are the best way to have asynchronous communication to external parties like deploy process or client adapters. VDS uses an event mechanism to notify the registered clients about any change in the vocabulary data. FIG. 4H is a block diagram of a class hierarchy that may be used to implement an event mechanism, in one embodiment.

3.0 INFORMATION OBJECT REPOSITORY (IOR)

According to one embodiment, the concept application 408 is an information object repository application. An information object repository (IOR) holds content for documents. For example, in this embodiment, the marketing document described above at URL address 'http:///www.Enterprise.com/Hello/Chap2/' is in the IOR. The content is stored and retrieved in units of data herein called information chunks. An IOR application produces documents, such as operating manuals, marketing documents, and Web pages for a Web site by combining one or more information chunks in the IOR. One or more IOR processes employed by the IOR application manage the IOR by relating the content in the IOR to one or more concepts in the concept database 420 and determine the information chunks to incorporate into documents based on one or more relationships in the concept database 420.

Using this technique, content originally unrelated and authored over time by many different persons and organizations can be related using the business vocabulary concepts and relationships in the VDS. Thus a person wishing to learn about the BetaPerseus 2.0 can use an IOR application to find all the manuals, press releases, and articles that describe it no matter when or by whom the document was written, as long as the content is registered with the IOR.

As another example, a system put together by a joint venture can produce a system document that uses descriptions of the components originally written independently by the joint venture partners. In addition, the information chunks supplied to a requestor can be tailored to the person making the request, for example, by providing more technical information to a technical user than to a marketing user. Furthermore, information chunks can easily be reused in several documents. For example, an introductory paragraph for the BetaPerseus 2.0 written for a marketing document can be used in a press release, a data sheet, and the home page for the BetaPerseus 2.0 on the Web site of the enterprise.

Embodiments are described herein in the context of examples involving generation of electronic documents in the form of Web pages. Embodiments are applicable to generation of any form of electronic document, and are not limited to use with Web sites or Web pages.

3.1 IOR CREATION LAYER

One set of IOR processes are used to manage the registration of information chunks into the IOR and the concept database. This set of IOR processes and the data storage for the IOR comprise the creation layer of the IOR, herein designated IOR-C. FIG. 4B is a block diagram illustrating the IOR-C of the IOR according to one embodiment.

In this embodiment, the IOR processes are invoked through an interface 462 for the IOR-C 460. For example, an application programming interface of the IOR-C interface 462 is invoked by a content generation application 444. In another example, an IOR administrator performs administration of the IOR through an administrator user interface of the IOR-C interface 462. In other embodiments the IOR processes execute under control of a standalone IOR batch or user-interactive application.

The IOR-C interface 462 includes methods to access the business vocabulary development server (VDS) 410 of the enterprise through the concept access API 432. As shown in FIG. 4B, this embodiment of the VDS 410b has an external concept access API 432 which uses a concept cache server 440 to speed retrievals from the VDS 410b. The concept cache server 440 uses a cache memory to temporarily store a subset of the concepts and relationships in the concept database of the VDS 410b.

The IOR-C interface 462 includes methods to store and retrieve information chunks in a content management system (CMS) such as in a local CMS 452 or over the network 401 in a remote CMS 458. A CMS includes persistent storage where an information chunk is stored. For example, persistent content store 454 includes information chunk 464.

A CMS is capable of managing a variety of types of information in each information chunk. For example an information chunk may comprise a block of text, an application

program, a query for a database, a vector graphic, an image, audio data, video data, and other binary data. The block of text may be text that represents code for a compiler, such as C code, and formatted text, such as text in the Hypertext markup language (HTML) or in the extensible markup language (XML), as well as unformatted text using one of several character codes, such as ANSI one byte and Unicode four byte codes.

In some embodiments, the CMS comprises the local operating system directory structure. For example, different information chunks are simply kept in different files with different file extensions for the different types of data, and the files are organized into one or more directories in a hierarchy of directories and files. In another embodiment, the CMS is a database server for managing a database of information chunks.

It is not necessary that all the information chunks be in a single CMS on one computer device. Data integration tools 456 are commercially available for associating data in one CMS, such as CMS 452, with data in another CMS, such as remote CMS 458. Appropriate data integration tools also can associate data that is in any other location that can be referenced, i.e., any object that exists, whether it is in a CMS or not, i.e., LDAP directories, Web services, application versioning, network addresses from DNS, physical objects such as bar codes, etc. In the depicted embodiment, the methods of the IOR-C interface access the data integration tools 456. In an embodiment with all the information chunks stored in a single local CMS, the data integration tools 456 are not included, and the methods of the IOR-C interface access the local CMS 452 directly.

Each information chunk in the CMS is identified uniquely by an information chunk reference 466. Depending on the CMS employed, the reference may be a file name, a file name including one or more directories in the hierarchy of directories, a network resource address, a universal resource locator (URL) address, a record identification in a predetermined database, or a record identification in a predetermined content management system.

FIG. 4B also shows a process 470 for generating pages 480 for a Web site on Web server 402 using the IOR-C interface to access the VDS 410 and the persistent content store 454. The process 470 is described in more detail in a later section.

The IOR-C interface 462 includes methods to manage the IOR by relating the information chunks in the CMS to one or more concepts in the concept database 420. The IOR-C interface includes methods to generate and retrieve information object concepts in the concept database associated with the information chunks. The IOR-C interface also includes methods to generate and retrieve relationships between the information object concepts and other concepts in the concept database.

3.2 INFORMATION OBJECTS AND RELATIONSHIPS

For each information chunk that is registered in the IOR 460 by a method of the IOR-C interface 462, a particular information object concept is added to the concept database of the VDS 410b. In one embodiment, an information object category is added to a Vocabulary Table. The particular information object is a child of the information object category and is represented as a new row in an Information Object Table. The concept cache server 440 or concept access API 432 is invoked by the IOR-C method to add this concept to the database.

Table 1 lists sample entries in a hypothetical Information Object Table according to this embodiment. In this embodiment, the information object concept has a name that is the unique reference for the corresponding information chunk in the CMS. As shown in Table 1, the unique reference is a URL in this embodiment.

Table 1. The Information Object Table

Name	Description	Creation Date
http://www.Enterprise.com/literature/ devices/catalog/Chap2/	marketing document for Perseus routers	9/19/00
http://www.Enterprise.com/ Hello/Chap2/	marketing document for Perseus routers	9/20/00
ftp://Enterprise.com/literature/ devices/Perseus/Intro17.txt/	BetaPerseus introductory paragraph for silver partner marketing person	12/12/00
ftp://Enterprise.com/literature/ devices/Perseus/Intro5.txt/	BetaPerseus 2.0 introductory paragraph for technical person	4/12/00
http://Enterprise.com/datasheets/DS33/	BetaPerseus 2.0 data sheet table	4/12/00
http://Enterprise.com/datasheets/DS12/	Jason data sheet table	4/12/00

Adding the information object concept to the concept database links the concept database to an information chunk in the CMS, but this action alone does not relate one information chunk to another. Once a particular information object concept has been added to the concept database, a relationship is formed with one or more other concepts in corresponding other hierarchies. As each information chunk has an information object concept added to the database and linked to another concept, relationships between the information chunks are implied by the relationships between the corresponding concepts.

For example, an instance of a "has info object" relationship type is added to the database to relate at least one product concept to each information object added. These relationships relate the first and second information objects in Table 1 to the Perseus concept in the product hierarchy, the third information object to BetaPerseus concept, the fourth and fifth information objects to the BetaPerseus 2.0 concept, and the sixth information object to the Jason concept in the product hierarchy. Since both the fourth and fifth information objects are related to the same product concept, by implication the information objects are related to each other. In this way, the information chunks referenced by URLs "ftp://Enterprise.com/literature/devices/Perseus/Intro5.txt/" and http://Enterprise.com/datasheets/DS33/ are related by implication, and can reasonably be placed in the same document in some circumstances. In this example, the two information

chunks referenced by the fourth and fifth information objects are closely related even though those two information chunks reside in the CMS in entirely different levels of different subdirectories in the Enterprise.com directory.

Similarly, since BetaPerseus 2.0 is a child of BetaPerseus and BetaPerseus is a child of Perseus in the product hierarchy, the corresponding information objects are implied to share this same hierarchical relationship. Thus one can navigate among the information objects, and their associated information chunks, using the relationships among the concepts related to the information objects. These techniques allow the rich collection of relationships in the concept database to provide organization for the information chunks stored in the CMS.

FIG. 5 is a block diagram that illustrates relationships involving a particular information object and other concepts in the vocabulary database according to another embodiment. According to this embodiment, meaningful documents are produced from information chunks by relating information object concepts not only to a concept in the product hierarchy but also to concepts in an information type hierarchy and concepts in a user profile hierarchy.

A particular information object concept 512 is a child of information object category 510 by the information object child of relationship indicated by arrow 511. The particular information object 512 is a participant in a 5-ary "has info object" relationship indicated by the five-pronged connection 590. This "has info object" relationship involves a concept 390 of the product hierarchy as in the above example, but also involves other concepts. The "has info object" relationship also involves a concept 542 in an information type hierarchy and two concepts in a "user profile" hierarchy, one a child concept 524 of a job role concept 522, and the other a child concept 532 of a partner level concept 530 . These other concepts and hierarchies are described in more detail next. The 5 participant "has info object" relationship specifies that a particular information chunk provides a particular information type about a

particular product in the product hierarchy of interest to a person playing a particular job role for a particular level of partner to the enterprise.

Introduction concept 542 is a child of an "info type" category 540 by the "info type child of" relationship indicated by arrow 541. According to this embodiment, various documents generated from the information chunks use or reuse one or more concepts of the "info type" category. The "info type" hierarchy is a one level hierarchy below the "info type" category as the root concept. Every different info type concept is a child of the "info type" root concept. The "info type" concepts include "Introduction," "Features and Benefits," "Product Photo," "Schematic Drawing," "Operational Properties," "Data Sheet Table," and "External Article Section," among others.

Each concept of the "info type" hierarchy is related to one or more concepts in a "document type" hierarchy by a "has_docinfo" relationship indicated in FIG. 5 by the double arrow 562. A "position" attribute of the "has_docinfo" relationship indicates where the information chunk corresponding to the information object is placed relative to information chunks corresponding to other information objects in the document. The "document type" hierarchy is a one level hierarchy below the "document type" category as the root concept. Every different document type concept is a child of the "document type" root concept. The "document type" concepts include "Marketing Document," "Product Home Page," "Data Sheet," "Press Release," "Operator's Manual," and "External Article," among others.

The "has_docinfo" relationship, such as 562, provides a specific organization of information chunks to produce a particular document of a given document type. For example, a product marketing document type is a participant in four "has_docinfo" relationships, one each with a "Product Photo" information type concept having a position attribute value "first," an "Introduction" information type concept having a position attribute value "second", a "Features and Benefits" information type having a position attribute value "third" and an "Ordering" information type having a position attribute value "fourth." For

another example, a product home Web page document type is a participant in five "has_docinfo" binary relationships, one each with a "Product Name Heading" information type in a first position, a "Introduction" information type in a second position, a "List of Other Info Types for Concept" information type in a third position, a "List of Other Documents for Concept" information type in a fourth position, and a "List of Next Concepts in Hierarchy" information type in the fifth position. The last three information type concepts do not need information objects to provide the content for the Web page document type, because these lists can be derived from the relationships in the VDS for a given concept. By tying information object concepts indirectly to document type concepts through information type concepts, a particular information chunk can be reused in more than one document.

Marketing role concept 524 is a child of a "job role" concept 522 which is a child of a "user profile" category 520 by the "user profile child of" relationship indicated by arrows 523a and 521a, respectively. According to this embodiment, the content in an information chunk for a particular information type for a particular product depends on the job role of the person who is reading the document. The content is expected to be different for a person in a marketing role, concept 524, than one in a technical role, concept 526, or even one in a sales role, concept 528, which may represent some mixture of the content provided for the two other roles. These other job roles are also children of the job role concept 522. Still other job role children concepts are defined in other embodiments. For example, a "default" concept child of the "job role" concept 522 is used for a person who does not play a particular role. This person is treated as an uninitiated member of the general public.

Silver partner concept 532 is a child of a "partner level" concept 530 which is a child of a "user profile" category 520 by the "user profile child of" relationship indicated by arrows 531a and 521b, respectively. According to this embodiment, the content in an information chunk for a particular information type for a particular product depends on the kind of partner to the enterprise is the corporate employer of the person who is reading the

document. A gold partner, represented by concept 534, is an entity or affiliate treated as any other member of the enterprise itself. A silver partner, represented by concept 532, has some favorable access to information. A default partner, represented by concept 536, has no formal relationship with the enterprise and is treated as a member of the general public. The content is expected to be different for partners at the different levels. These particular partner levels are all children of the "partner level" concept 530. Still other children concepts are defined in other embodiments.

Table 2 gives the entries in the relationship type table for the relationship depicted in FIG. 5 by connection 590. Table 3 gives the entries in the participant type table for the relationship depicted in FIG. 5 by connection 590.

Table 2. The "Has Info Object" Entries in Relationship Types Table

Relationship Type Name	Description	Number of Participants	Creation Date
has_info_object	references content	5	4/12/2000

Table 3. The "Has Info Object" Entries in Participant Types Table

Relationship Name	Role	Participant Type
has_info_object	info_object	info_object
has_info_object	info_type	info_type
has_info_object	concept	Vocabulary
has_info_object	job_role	job_role
has_info_object	partner_level	partner_level

The particular instance of the "has info object" relationship depicted in FIG. 5 specifies that a particular information chunk referenced by a particular information object 512 provides an "Introduction" info type about the BetaPerseus 2.0 product of interest to a person playing a marketing role for a silver level partner to the enterprise.

To manage a plurality of information chunks, the IOR-C interface provides methods for defining the information object type, for setting the attributes of a concept of the

information object type, for defining the "has info object" relationship type involving the information object type, and for setting the attributes of a particular relationship of the type.

5.3 METHOD OF MANAGING INFORMATION OBJECTS

FIG. 6A is a flow chart illustrating a method 601 for managing an information object repository by generating and storing an information object according to one embodiment.

In step 602, a method of the IOR-C interface receives an information chunk, such as a block of text. In step 604, the information chunk is sent to the content management system (CMS) for storage and a unique identification for the chunk is returned by the CMS to use as a reference for retrieving the information chunk from the CMS. For example, the reference may be the URL of a file in which the information hunk is stored.

In step 606, the method of the IOR-C invokes a method of the concept access API or the concept cache server to instantiate a particular information object of the information object category with a name of the unique reference. A description attribute of the particular information object is set to a text string that describes the information in the information chunk or left blank.

In step 608 the information chunk is associated with a concept in the concept database, an information type, a job role and a partner level. In one embodiment, a user, such as a subject matter expert, is prompted for this information. In another embodiment, the information is provided with the information chunk itself. In yet another embodiment this information is derived from information provided with the information chunk or in the information chunk by the content generation application (444 in FIG. 4B). In one embodiment, the description attribute of the particular information object concept is edited to reflect this information.

In step 610 the method of the IOR-C interface invokes a method of the concept access API or the concept cache server to instantiate a particular relationship of the "has info object" relationship type in the concept database of the VDS 410.

FIG. 6B is a flow chart illustrating a method 620 of the IOR-C interface for managing an information object repository by retrieving an information object according to one embodiment.

In step 622 the method receives a request for a particular information type for a particular concept. For example, the request includes data indicating an "Introduction" information type is desired for the "BetaPerseus 2.0" product.

In step 624 the method generates a string naming the relationship type that has an information object as a participant. In this example that string contains the relationship type named "has_info_object."

In step 626 the method of the IOR-C interface invokes a method of the concept access API or the cache server to get participants of each instance of the relationship involving the concept. For example, the method of the IOR-C interface invokes a method named "getParticipants" of the concept access API to get participants of each instance of the "has_info_object" relationship involving the concept "BetaPerseus 2.0." In step 628 the method receives an array of strings giving the relationship instance identification (rID) and the participant concepts in the relationship instance. For example, the array of three instances of the "has_info_object" relationship listed in Table 4 is received.

Table 4. Example Instances of "has_info_object" Relationship Returned

RID	Role	Participant
1117	info_object	ftp://Enterprise.com/literature/devices/Perseus/Intro5.txt/
1117	info_type	Introduction
1117	concept	BetaPerseus 2.0
1117	job_role	Technical
1117	partner_level	Default
4567	info_object	http://Enterprise.com/datasheets/DS33/
4567	info_type	Data Sheet Table
4567	concept	BetaPerseus 2.0
4567	job_role	Default
4567	partner_level	Default
9877	info_object	ftp://Enterprise.com/literature/devices/Perseus/Intro27.txt/
9877	info_type	Introduction
9877	concept	BetaPerseus 2.0
9877	job_role	Marketing

9877	partner_level	Silver

In step 630, this list is filtered to remove those relationships not involving the information type indicated in the request. For example, the relationship instance with rID equal to 4567 is removed because it is not an "Introduction" information type concept specified in the example request. In step 632 the filtered instances of the relationship are returned to the requesting process. For example, the array of Table 16 is returned with the lines for rID of 4567 absent.

FIG. 6C is a flow chart illustrating a method 640 of the IOR-C interface for managing an information object repository by retrieving information content associated with an information object according to one embodiment. In step 642 the method receives a request from a user for the information chunk for a particular information type and a particular concept. For example, a user who plays a marketing role with a silver partner requests the information chunk associated with the introduction to the BetaPerseus 2.0 product.

In step 644 the method 640 invokes a method to return the participants in the relationship instances involving an information object concept, the specified concept, and the specified information type concept. As a result, an array of participants for any instances of such a relationship is returned. For example, the method 640 invokes the method 620 to return the participants in the "has_info_object" relationship instances involving the "BetaPerseus 2.0" concept and the "Introduction" information type concept. As a result the array listed in Table 16, excluding the rows with rID of 4567, is returned.

In step 646, the returned array is filtered to remove instances that do not match the user associated with the request. For example, the rows of Table 16 having an rID of 1117, which involve the technical job role, are eliminated because the user associated with the request is a marketing person, not a technical person. Only the rows with an rID of 9877 remain.

In step 648, the references to the information chunks are taken from the particular information object participants in the filtered instances. For example, the URL "ftp://Enterprise.com/literature/devices/Perseus/Intro27.txt/" is obtained from the information object participant in the only remaining relationship instance, the instance having rID 9877.

In step 650, the method of the IOR-C interface requests the information chunk having the reference from the content management system. For example, the IOR-C interface requests from the CMS the information chunk having the URL "ftp://Enterprise.com/literature/devices/Perseus/Intro27.txt/." In step 652, the retrieved information is received and returned to the requesting process.

As described above, the IOR-C interface provides methods for storing information content, for generating and storing an information object associated with the information content, for retrieving an information object, and for retrieving the information content associated with an information object. As described below, other layers of the IOR are generated and used with other interfaces to support fast, dynamic document production based on the concepts and relationships in the vocabulary development server (VDS) and the content in the content management system (CMS).

5.4 MULTIPLE LAYER IOR

FIG. 7 is a block diagram illustrating an information object repository management layer 782a (IOR-M), a staging layer 782b (IOR-S), and a Web server layer 782c (IOR-F) of a multiple layer information object repository according to one embodiment. Such layers of the IOR are generated and used with IOR layer interfaces 784a, 784b, to support dynamic electronic document production based on the concepts and relationships in the VDS and the CMS.

Transform process 772 obtains information from the IOR-C layer 460 using the IOR-C layer interface 462 and generates the IOR-M layer 782a using the IOR-M layer interface 784a. The IOR-M layer 782a includes a content cache 778a, a concept cache 774a, and a

concept cache server 740a. A management layer tool 773 also uses the IOR-M interface 784a to allow a user to view and edit the information chunks, concepts and relationships in the IOR-M 782a.

The content cache is a data store that includes a subset of the information chunks stored in the CMS of the IOR-C layer. Information chunks that have become obsolete or that are not yet released are excluded from the content cache 778a. Also excluded are information chunks that are not used by the electronic documents to be produced. Some information chunks of the CMS are combined into a single chunk in the content cache 778a, if doing so is expected to enhance efficiency of use. For example, information chunks that are always used together in the documents to be produced may be combined into a single information chunk in the content cache 778a.

The concept cache is a data store that includes a subset of the concepts and relationships stored in the concept database. The concept data from the concept database is de-normalized in the concept cache to improve speed of retrieval by allowing a concept that participates in more than one relationship to be stored more than once in the concept cache. For example, an information type concept is stored with other information type concepts in the information type hierarchy and again with each document type that includes the information type. Thus, when a document is generated, a full description of the information types is with the document type, reducing time needed to retrieve such data from the concept cache.

The management layer concept cache server 740a provides access to the concepts and relationships in the management layer concept cache 774a. The cache servers 740 in all the layers support the same methods provided by the concept access API, but each cache server 740 operates on the concept cache 774 in the same layer.

The deploy process 774 obtains information from the IOR-M layer 782a using the IOR-M layer interface 784a and generates the IOR-S layer 782b using the IOR-S layer

interface 784b. The IOR-S layer 782b includes a second content cache 778b, a second concept cache 774b, and a second concept cache server 740b. A quality assurance application, such as the model electronic document generator 775, uses the IOR-S interface 784b to allow a user to test the IOR-S layer 782b for its suitability for generating documents to be provided in a later stage. For example, the IOR-S layer is tested using the IOR-S interface 784b to ensure that all information chunks in the content cache have an information object concept in the concept cache and that the information object concept has a relationship with at least an information type concept. As another example, authors use the IOR-S interface 784b to view the information chunks and determine that the information is correct for the concepts to which the information chunks are related by a corresponding information object.

A model electronic document generator 775 also uses the IOR-S interface 784b to allow a Web site developer to generate, view and edit the electronic documents to be provided by the Web server. In the course of operations the model electronic document generator 775 produces a search index 787a, a directory structure 788a for storing electronic documents produced, and static electronic documents 789a that do not depend on the user profile of the user viewing the page.

The replicate process 776 obtains information from the IOR-S layer 782b and reproduces it in a fast, Web server layer, IOR-F, 782c on each of one or more Web servers, such as Web server 402. The IOR-F layer 782c includes a third content cache 778c copied from the second content cache 778b, a third concept cache 774c copied from the second concept cache 774b, and a third concept cache server 740c. The search index 787a, directory structure 788a and static pages 789a are also replicated as search index 787b, directory structure 788b and static pages 789b, respectively, on each of the Web servers, such as Web server 402.

An electronic document generator 786 produces electronic documents 480 in response to requests from client 404. The electronic document generator 786 uses the search index 787b, the directory structure 788b, the static pages 789b, and the IOR-F layer 782c in any combination to produce the electronic documents 480. As described in more detail in the next section, when a user selects a concept with content that depends on the user profile, the electronic document generator uses the concept cache server 740c to determine information types in the electronic document type, and the information objects related to the information types for the selected concept. The electronic document generator then retrieves the information chunks from the content cache using the URL reference from the information object.

Also shown on Web server 402 is a web application 790 that may be requested through the electronic documents 480.

In the following sections, the use of the IOR-F layer is first described to illustrate the dynamic document production to be supported by the multiple layer IOR. Then the use of the IOR-M and IOR-S layers are described to show how those layers support the formation of the IOR-F layer.

5.5 IOR WEB SERVER LAYER (FAST LAYER)

FIG. 8A is a flow chart illustrating a method 810 for generating a static electronic document based on the IOR-F layer 782c according to one embodiment.

In step 812, a user profile is obtained, such as when a client 404 operated by a user contacts the Web server 402 and logs on. A default user profile is used if the Web server does not require or provide a user log-in process. A list of concept categories available through the site is presented on a static electronic document returned to the client. The static electronic document is found using the directory structure 788b of the Web server 402.

FIG. 8B is a flow chart illustrating a method 820 for generating a concept home electronic document based on the IOR-F layer according to one embodiment.

In step 822 data is received from the client process indicating a concept selected by the user. The concept selected is the concept whose home electronic document is to be produced. For example, the data may indicate the user has selected a Product root concept so that a Product root concept home Web page is to be produced. In an alternative example, the data may indicate the user has selected the BetaPerseus 2.0 concept so that a BetaPerseus 2.0 home Web page is to be produced. In one embodiment, the electronic document generator finds the information types that constitute a concept home page document from the concept cache server and finds that it includes five information types, as listed above; namely, a "Product Name Heading" information type in a first position, an "Introduction" information type in a second position, a "List of Other Info Types for Concept" information type in a third position, a "List of Other Documents for Concept" information type in a fourth position, and a "List of Next Concepts in Hierarchy" information type in the fifth position.

In step 824, the information chunks for the page are retrieved. For example, the information chunk associated with the "Introduction" information type for the BetaPerseus 2.0 concept for the job role and partner level in the user profile of the user is retrieved. In one embodiment the method 640 illustrated in FIG. 6C is invoked from the concept cache server 740c in the IOR-F layer.

In another embodiment, the concept cache server 740c determines the information object related to the concept, information type and user and returns the URL of the information chunk from the information object, and the electronic document generator retrieves the information chunk based on the URL returned. For example the concept cache server 740c returns the URL "ftp://Enterprise.com/literature/devices/Perseus/Intro27.txt/" and the electronic document generator retrieve the information chunk stored at that URL.

In the example embodiment, the Product Name Heading information type is based on the concept name and does not require an information chunk be retrieved. Similarly, the list

of other information types and other documents for the concept and user depend on information in the concept cache and also do not require an information chunk be received.

In step 826 the other information types for this concept that have information objects are retrieved from the concept cache server 740c based on the concept cache 774c to supply the list of other information types for the electronic document. The user may later be allowed to retrieve any of the information chunks for the information types listed on this page. For example, the other information types that have information objects for BetaPerseus 2.0 for the silver partner marketing user are returned, such as a "Data Sheet Table" and a "Features and Benefits" information type. If an information object is not related to an information type, that information type is not listed. For example, if a "Product Photo" is not available for this product for this user, then the "Product Photo" is not included in the list of available information types.

In step 828 the other document types for this concept that have information objects are retrieved from the concept cache server 740c based on the concept cache 774c to supply the list of other documents for the electronic document. The user may later be allowed to retrieve any of the documents listed on this page. For example, the other document types that have information objects for BetaPerseus 2.0 for the silver partner marketing user are returned, such as a "Data Sheet" and a "Marketing Document" document type. If an information object is not related to every information type of a document type, that document type is not listed. For example, if a "Press Release" is not available for this product for this user, then the "Press Release" is not included in the list of available document types.

In step 830 an electronic document 480 is generated for the concept home electronic document and sent to the client 404. This step includes finding the next concepts in the hierarchy by requesting them from the concept cache server. The concept home electronic document comprises a concept name as a title, information from the information chunk associated with the "Introduction" information type, a selectable list of other information

types, a selectable list of other document types, and a selectable list of the next concepts in the hierarchy. For example, the BetaPerseus 2.0 home electronic document includes the title "BetaPerseus 2.0", the Introduction information chunk stored in URL "ftp://Enterprise.com/literature/devices/Perseus/Intro27.txt/", a list of other available info types including "Data Sheet Table" and "Features and Benefits," a list of available document types including "Data Sheet" and "Marketing Document," and a list of the next concepts in the Product hierarchy, "BetaPerseus 2.4" and "BetaPerseus 3.0."

FIG. 8C is a flow chart illustrating a method 840 for generating an information type electronic document for a concept based on the IOR-F layer according to one embodiment. This method is invoked, for example, if the user selects the "Features and Benefits" information type on the BetaPerseus 2.0 home Web page.

In step 842, the method receives data indicating a user selection of a particular information type for the concept. In step 844, the method uses the concept cache server to find the information object related to the concept, user, and information type. The concept server cache returns the information object identified by the unique URL reference to the information chunk in the content cache. For example, the concept cache server is used to find the information object related to a "Features & Benefits" information type for the "BetaPerseus 2.0" concept for a marketing role person of a silver partner. The concept server cache returns an information object identified by the URL, such as "ftp://Enterprise.com/marketing/FandB44.txt/", the unique reference to the information chunk in the content cache.

In step 846, the URL returned to the electronic document generator is used to retrieve the information chunk. For example, the information chunk in "ftp://Enterprise.com/marketing/FandB44.txt/" is retrieved. In step 848, an electronic document is generated that includes the information chunk, and the electronic document is sent to the client process. For example an electronic document 480 showing the features and

benefits of the BetaPerseus 2.0 of interest to a marketing person of a silver partner is produced and sent to client 404.

FIG. 8D is a flow chart illustrating a method 860 for generating a concept document electronic document based on the IOR-F layer according to one embodiment. This method is invoked, for example, if the user selects the "Data Sheet" document type on the BetaPerseus 2.0 home Web page.

In step 862, the method receives data indicating a user selection of a particular document type for the concept. In step 864 the method uses the concept cache server to find the information types included in this document type. For example, the Data Sheet document type includes the "Introduction" information type and the "Data Sheet Table" information type. In step 866, the method uses the concept cache server to find the information objects related to the concept, user, and information types. The concept server cache returns the information objects identified by the unique URL references to the information chunks in the content cache. For example, the concept cache server is used to find the information objects related to an "Introduction" information type and a "Data Sheet Table" information type for the "BetaPerseus 2.0" concept for a marketing role person of a silver partner. If a information object is not available for the specific role or partner a default role or partner or both is used in this embodiment. The concept server cache returns information objects identified by their URLs, such as "ftp://Enterprise.com/literature/devices/Perseus/Intro25.txt/" and "http://Enterprise.com/datasheets/DS33/", the unique references to the information chunks in the content cache.

In step 868, the URLs returned to the electronic document generator are used to retrieve the information chunks. For example, the information chunks in "ftp://Enterprise.com/literature/devices/Perseus/Intro25.txt/" and "http://Enterprise.com/datasheets/DS33/" are retrieved. In step 870, an electronic document

representing the document is generated that includes the information chunks, and the electronic document is sent to the client process. For example a Data Sheet electronic document showing the Introduction for the BetaPerseus 2.0 and a Data Sheet Table for the BetaPerseus 2.0 of interest to a marketing person of a silver partner is produced and sent to client 404

FIG. 8E is a flow chart illustrating a method 880 for generating a concept search result electronic document based on IOR-F layer according to one embodiment. This method is invoked, for example, if the user inputs a key word into a "search" field on the static home page.

In step 882, the method receives the search term from a particular user having a user profile, such as a visitor profile. In step 884, the electronic document generator uses the search index 787b to find the search term. The index 787b provides a list of information objects in the concept cache for each term. Use of index 787b is one option to translate a search query that may have alternative forms. Alternatively, such translation may occur through one or more programmatic function calls to methods of an application programming interface. In step 886, if the search term is found in the index, then the information objects listed for the found term are retrieved by the electronic document generator.

In step 888, the concept cache server is used to determine the concepts and users related to the found information objects, and the information objects are filtered to remove the information objects related to users that do not match the user profile of the user requesting the search. In this context, an information object for a default job role or default partner match the role and partner, respectively, of the user making the request.

In step 890, the electronic document generator produces an electronic document displaying the search term and listing the concepts related to the filtered information objects, and for each concept lists the names of the filtered information objects related to the concept.

In an alternative embodiment, the information types related to the filtered information objects are listed in lieu of or along with the names of the filtered information objects.

For example, a search on the term "SuperPerseus" may find the term in the index along with a particular information object associated with the information chunk containing the section of the article that coined the term. If the particular information object were generated and related to users in default job roles and default partner levels in the IOR-C level, then the electronic document produced would display the search term "SuperPerseus" and display the concept "BetaPerseus 2.5" and list the information object, such as "http://TechJournal.com/V9/Issue11/article2/", and an information type, such as "External Article Section."

FIG. 8F is a flow chart illustrating a method 891 for generating an information chunk electronic document based on the IOR-F layer according to one embodiment. This method is invoked, for example, if the user selects the "http://TechJournal.com/V9/Issue11/article2/" information object on the search results electronic document.

In step 892, the electronic document generator receives data from the client 404 indicating the user's selection of a particular information object. In step 894, the electronic document generator retrieves the information chunk stored in the content cache at the URL address of the information object. In step 896, the electronic document generator produces an electronic document displaying a concept name, an information type name, and the information chunk and sends the electronic document to the client process 404 for display to the user. This step may include retrieving the concept and information type related to the information object from the concept cache server. For example, if the user selects the "http://TechJournal.com/V9/Issue11/article2/" information object on the search results electronic document, the user is then presented with an electronic document showing "BetaPerseus 2.5" "External Article Section" and one or more formatted paragraphs from the article coining the phrase "SuperPerseus."

Using the IOR-F layer, an electronic document generator can produce dynamic electronic documents tailored to a particular user based on content in the content cache, arranging the content on the electronic document based at least in part on data in the concept cache.

5.6 IOR MANAGEMENT LAYER

FIG. 9A is a flow chart illustrating an embodiment 772' of a method for generating and managing the IOR-M layer. In the embodiment described, these steps are performed by an embodiment of the transform process 772 depicted in FIG. 7. In other embodiments, some of these steps are performed by a user or application invoking the management tool 773 in FIG. 7.

In step 902, data indicating a range of users, dates and categories of concepts are received that help determine which documents are to be generated in the IOR-F layer. For example, the data received may indicate that only documents about products and services will be produced in the IOR-F layer, not documents about research projects or joint ventures. This information helps determine the subset of concepts and content to be moved to the management layer.

In step 904, the next concept is retrieved from the IOR-C layer interface 462, which invokes a method of the concept access API (432 in FIG. 4B) or the concept cache server (440 in FGI. 4B). In the alternative embodiment shown, the transform process accesses the concept cache server 440 directly rather than through the IOR-C layer interface 462. In an embodiment using the management tool 773 the next concept is retrieved by invoking a method of the IOR-M layer interface 784a. The first time step 904 is performed a first concept retrieved from the IOR-C layer interface is the "next" concept.

In step 906, the information chunks associated with the concept are retrieved from the IOR-C layer interface 462. In one embodiment this involves invoking a method of interface 462 that returns all the participants in a "has_info_object" relationship with the concept.

Then, the unique reference in each info object participant is used to retrieve the associated information chunk.

Step 908 represents a branch point at which it is determined whether the information chunk is beyond the range of users, dates and categories of concepts that are the topics for the documents to be produced in the IOR-F layer. Step 908 also represents the branch point for determining whether the information chunk is obsolete or not yet released for distribution in documents to users. If all the information chunks are beyond the range or obsolete or pre-release, then control passes to step 910. Otherwise control passes to step 920.

In step 910 the concept is added to the management layer concept cache 774a as part of a hierarchy of concepts in a denormalized mode, such as by repeating the same concept for each relationship that involves the concept. Control then passes back to step 904 to process the next concept.

In step 920, the subset of information chunks that are not out of range or obsolete or pre-release are added to the management layer content cache 778a. The position of the information chunk is likely changed; therefore the reference that uniquely identifies the chunk for retrieval has likely changed also. In some cases two or more information chunks from the IOR-C layer are combined into a single information chunk for the management layer content cache. Each information chunk of the subset is stored with its new references, e.g. its new ID, in the content cache.

In step 922, the concept is added to the management layer concept cache 774a as part of a hierarchy of concepts in a denormalized mode. An information object concept is added using the new ID as a new reference. One or more relationships involving the information object and another concept are also added to the management layer concept cache.

In step 924 a search index is generated with a variety of terms from the information chunk. For each term in the information object a term is found in the index, or added to the index if not already there. Then the name of the information object is added to a list of

information objects for the found term, or starts the list of information objects for the added term. In many embodiments, the information object name is the unique reference to the associated information chunk. In other embodiments, the search index is not generated in the transform process 772 but is performed using the management tool 773. In still other embodiments, the search index is not generated in the IOR-M layer, but instead in the IOR-S layer.

After step 924, control passes back to step 904 to retrieve the next concept.

Using these techniques, streamlined subsets of the information chunks in the CMS and the concepts and relationships in the concept database of the VDS are formed and managed, including being viewed and edited, to expedite the dynamic production of documents.

5.7 IOR STAGING LAYER

FIG. 9B is a flow chart illustrating an embodiment 774' of a deploy method for generating and managing the IOR-S layer 782b. In step 944 the next concept is retrieved from the management layer concept cache server 740a using the IOR-M layer interface 784a. At the start of the process, the first concept retrieved is the next concept. When no concepts remain to be the next concept, the process terminates. In step 946 the associated information chunk is retrieved from the management layer content cache 778a through the IOR-M layer interface 784a. This step invokes a method that retrieves the information objects related to the concept in a "has_info_object" relationship and uses the unique references in the information objects to retrieve the information chunks.

In step 948 the information chunk is stored using the IOR-S layer interface in the staging area content cache 778b with a new unique reference in the form of a relative URL in a directory structure. In other embodiments, the URL is an absolute URL address for the intended storage when replicated to a IOR-Fast layer on one or more Web servers. In still other embodiments, the relative URL is assigned in a separate process, such as the model

electronic document generator process 775, after the information chunks are stored in the IOR-S layer 782b. In step 950, the current concept is stored in its concept hierarchy in the staging layer concept cache 774b and the information object with the new reference is also stored in the staging layer concept cache 774b using the IOR-S layer interface 784b.

If a search index was generated with the data in the management layer, that index is also copied in step 952, changing the information object name to reflect the new unique reference, such as the new information chunk ID, in some circumstances.

FIG. 9C is a flow chart illustrating an embodiment 775' of a method for preparing a Web site based on the IOR-S layer 782b. For purposes of illustrating a simple example, FIG. 9C is described herein in the context of preparing a Web site. However, the processes described herein are equally applicable to managing other information, e.g., sets of electronic documents relating to mobile services, Web services, print documents, etc.

In step 962 a model electronic document generating process generates and previews Web pages for the Web site, including forming a directory structure788a that can be replicated to each of the Web servers that host the Web site, and a search index 787a, if one has not yet been formed. The content for these model pages are obtained from the IOR-S layer using the IOR-S layer interface 784b for modifying the content and concept caches, and using the concept cache server to retrieve the relationships and concepts that define documents and that point to the information chunks in the content cache.

In step 964, static pages 789a for the Web site are generated. The static pages do not depend on the user profile of the user operating the client process 404 requesting the page. The content for these model pages are obtained from the IOR-S layer using the IOR-S layer interface 784b for modifying the content and concept caches, and using the concept cache server to retrieve the relationships and concepts that define documents and that point to the information chunks in the content cache.

In step 966, the IOR-S layer interface is used to determine that every information chunk in the content cache has an information object in the concept cache and is related to at least one other concept in the concept cache. An information chunk can be orphaned, e.g., left without an information object and relationship, if a concept or information object is deleted from the concept cache. If orphaned, an information object referencing the chunk and at least one relationship to another concept are added to the concept cache using the IOR-S layer interface.

In step 968, information chunks are viewed by authors or experts and edited if incorrect or insufficient for the information type, concept or user to which they are related, using the IOR-S layer interface.

In step 970, the computational resources employed to dynamically generate Web pages for the Web site are measured to determine if a new arrangement of concepts in the concept cache or information chunks in the content cache is worthwhile.

In step 972, the index entries in the search index are tested for currency and edited by authors or experts if no longer deemed appropriate for the edited content cache or concept cache.

Using these techniques, the content cache, concept cache, search index, directory structure, and static pages are formed, reviewed and honed in the IOR-S layer for supporting the correct, rapid, and dynamic production of Web page based documents.

Once generated, reviewed, and edited as deemed fitting, the content cache, the concept cache, and the concept cache server, the static pages, the search index, and the directory structure of files are replicated to one or more Web servers, as indicated in FIG.7 by the replicate process 776. These replicated structures provide content and functionality to a Web page generating process installed on each of the one or more Web servers. The content cache, the concept cache, and the concept cache server constitute the IOR Web server layer, i.e., the IOR fast layer, IOR-F, 782c. The content cache provides content to the Web

page generating process. The concept cache and concept cache server provide the arrangement of the content to the Web page generating process. The number and location of Web servers are determined by load balancing considerations using any method known in the art at the time of the replication.

4.0 CACHE FOR INFORMATION OBJECTS INCLUDING VOCABULARY INFORMATION

According to an embodiment, a high-performance, multi-threaded, distributed caching system is disclosed that can process a high volume of client requests for accessing and querying concepts, relationships and information objects. In one specific embodiment, the caching system features a unified tree data structure that represents vocabularies, relationships and information objects in memory. The information objects serve as proxies to actual stored information chunks. Persistent storage is provided for storing the in memory data structure. A de-normalized table facilitates fetching information objects faster, and features caching the information objects using a least-recently-used algorithm. The information objects may comprise vocabulary information, attribute information, relationship information, etc. Thus, a caching system is provided that features high performance, optimum memory use by in memory data structure, support for large number of simultaneous requests, intelligent caching strategies at deployment, layers across distributed tiers and systems, scalability, stability, and adaptability to changing requirements.

FIG. 11A is a block diagram of an example embodiment of a cache system.

A client 1104 is communicatively coupled to a network 1101. At a server side of the network 1101, input data from subject matter experts (represented by block 1114) is received into the vocabulary development server 410. In this context, subject matter experts are individuals associated with the enterprise that owns or operates vocabulary development server 410 and who have knowledge about particular products, services, solutions, technologies, or other information using vocabulary terms that are managed by the

vocabulary development server, or managed by associated tools or processes that interface with application programming interfaces of the vocabulary development server. Thus, the subject matter experts act as arbiters of what vocabulary terms are correct, alternatives, incorrect, etc., and store such information in the vocabulary development server 410. The subject matter experts may do so by processing through workflow steps and actions that are based on their access level and other relationships to the concepts, attributes and relationships.

A concept cache server 440 is communicatively coupled to vocabulary development server 410, and the cache server also has access to tables in persistent content store 454, which comprise persistently stored cache data.

The cache server 440 communicates with external applications and servers through a concept access application programming interface (API) 432. Calls to the API and results from the API are provided to an interface 1112 to a delivery engine library. A preview server acting as a delivery engine, represented by block 1110, is communicatively coupled from the interface 512 to network 401. The delivery engine 1110 dynamically generates electronic documents in response to requests of client 1104, and can allow updates to system information from clients. In particular, the delivery engine 1110 determines a format of an electronic document that is responsive to the client request, and queries concept cache server 440 through concept access API 432 to obtain one or more information objects that form components of the dynamically generated electronic document. The information objects provide all things required to generate the document, including content, applications, images, templates, code, etc. When the electronic document is fully assembled, the delivery engine 1110 delivers the document to the client 1104 through network 1101.

Use of interface 1112 and delivery engine 1110 improves scalability by enabling the cache server 440 to respond to a large number of clients 1104 wherein data presentation and delivery are controlled by delivery engine 1110. Further, this arrangement enables concept

cache server 440 to cache information objects, vocabulary trees, relationship instances, or other components of electronic documents that are dynamically generated by the delivery engine 1110. Unlike past approaches that involve caching static pages, delivery engine 1110 can search the cache server 440 to obtain needed information objects in the process of assembly a dynamically generated electronic document. If the requested information objects are not in the cache of the cache server 440, the cache server can query the persistent content store 454 to obtain the requested objects. The cache server also can query any other logical store or network resource to obtain the requested objects, e.g., an LDAP directory, database server, etc.

Further, delivery engine 1110 is not required to interact directly with the persistent content store 454 or any other repository. Unlike past approaches, delivery engine 1110 is not required to carry out operations with the content store that involve substantial overhead, such as establishing a database connection each time that the delivery engine needs a particular information object, querying the database, finding the correct object, retrieving the information objects from relatively slow disk storage of the database, etc. Such queries would be required frequently, because a complete page would become invalidated often as a result of changes in data underlying one or more information objects that make up the page. By caching individual information objects rather than entire pages, a substantial performance bottleneck is avoided. Further, because all the vocabularies and information objects are connected, the system can determine which pages use a particular information object and use such data to rebuild only those pages.

Use of a vocabulary tree as defined herein in combination with the information object model defined herein introduces dependencies among objects, which in turn allows for predictive caching and query optimization. Predictive caching involves caching an object based on an expectation of what information a user will next request, based on the information that has been requested.

Arranging the cache server at approximately the same logical layer as the delivery engine also results in improved performance because socket calls or other relatively slow communication mechanisms are not required between the delivery engine and the cache server. Instead, the delivery engine can directly call the cache using programmatic calls. If the cache needs to obtain different information objects from the database, then socket calls may be used, but this arrangement pushes the slower connection mechanism lower in the logical structure so that its performance impact is reduced.

FIG. 11B is a block diagram of a second example embodiment of a cache system. In this embodiment, a plurality of cache servers 440A, 440B, 440N, etc., are distributed over a geographical area. Each of the cache servers 440A, 440B, 440N, manages a corresponding replicated persistent content store 506A, 506B, 506N at its location. A distributed cache server manager 1116 is communicatively coupled to all the cache servers 440A, 440B, 440N, through one or more intermediary networks, and to the preview server 1110. The distributed cache server manager 1116 manages the cache servers 440A, 440B, 440N, and directs client requests to them.

In this configuration, the cache system provides a stable system, using a distributed model that allows not only scalability but also load balancing and fail-over.

FIG. 12 is a block diagram that illustrates in more detail an internal arrangement of one of the cache servers 440. To facilitate an understanding of FIG. 12, a high-level overview of its structure and function is first presented, followed by a more detailed operational example.

In general, FIG. 12 depicts an arrangement that facilitates rapid retrieval of information objects from a cache, using an indexing method that is optimized for use with hierarchical trees of information objects, and n-ary relationships among such objects, as disclosed herein. In this arrangement, a hierarchical tree of information objects is viewed as a flat table comprising a plurality of columns. However, because of the potential size of the

object trees—which may contain on the order of tens of millions of information objects—a memory optimization arrangement is provided that reduces the amount of memory needed to store the flat tables. In particular, a way of traveling through the nodes of the trees is provided, by carrying out a lookup and then caching the result set. In prior approaches, a database table lookup returns a result set of rows, and those rows and their associated column values are cached "as is." In the approach herein, only the index values of result set rows are cached. Since the index values are numeric, the amount of memory needed to cache result sets is greatly reduced by omitting large or complex column values that are associated with rows in the result sets. Thus, in the disclosed arrangement, minimum memory usage and rapid retrieval are achieved.

Requests for information objects are received from a client 1202 at a query processor 1204. Each client request is presented in the form of a query. For example, a query might encapsulate the request, "Show me an electronic document that describes the Features & Benefits of Product 7500." In this context, client requests may be issued programmatically from one or more software applications that are executed by client 1202. For example, a client request may be formulated and generated by a Web application that the client uses to interface to a complex enterprise Web site.

The example query above involves three database columns: the category Product; the concept "7500"; and the information type "Features & Benefits." The desired result of the query is an information object of type "Features & Benefits" that matches the concept "7500" and the Product category.

Cached content 1214 provides basic cached storage of vocabulary information relating to information objects that may satisfy the client request. In one embodiment, cached content 1214 is structured as a flat table having a plurality of rows. Each row has column values that comprise a row identifier, a concept value, an information type value, and an index pointer value. In one embodiment, the index pointer values reference specific

information objects among a plurality of stored content chunks 1216. The stored content chunks 1216 are organized according to the hierarchical tree structure described herein with reference to FIG. 3 or FIG. 5. Each such hierarchy and its relationships may be represented in memory of a computer system as a logical tree using object-oriented programming techniques, such that each node of the tree is an object with attributes and relationships.

Stored content chunks 1216 serve as a local cache for information objects that are persistently stored in object repository 1220. Stored content chunks 1216 may be stored in a file system, database, etc. Least-recently-used information objects in stored content chunks 1216 are removed according to a conventional LRU algorithm. Use of an LRU process ensures that the amount of memory required by stored content chunks 1216 remains within a specified reasonable limit.

Additions, deletions, and changes to stored content chunks 1216 are received from a vocabulary index builder 1216 that communicates with cached content 1214. The vocabulary index builder is manipulated by an administrative user who can retrieve master copies of content objects from an object repository 1220 through a database access API.

Cached content 1214 is managed by software elements that ensure that the least-recently-used (LRU) cache entries are regularly deleted from the cache. This controls the cache size.

In operation, query processor 1204 parses a client request. Query processor 1204 then searches result cache 1206 to determine whether a result set value is present that matches the concept and information type in the query. The result set identifies one or more rows in cached content that contain index pointers to information objects in stored content chunks 1216, or in object repository 1220, that are responsive to the user query. If a result set is present, then the information objects that are referenced in the result set index pointers are obtained from either stored content chunks 1216, or object repository 1220, and returned to client 1202.

If result cache 1206 does not contain a matching result set, then cached content 1214 is searched according to the following process. Query processor 1204 generates and sends one or more index lookup requests to cached content 1214. In response to the lookup requests, cached content 1214 is searched and one or more interim result sets, represented by first interim results 1212A and second interim results 1212B, are generated.

Each set of interim results comprises a table of rows having a plurality of column values. In first interim results 1212A, the column values comprise a concept identifier, and a set of row identifiers for rows of cached content 1214 that match the associated concept identifier. In second interim results 1212B, the column values comprise an information type value, and a set of row identifiers for rows of cached content 1214 that match the associated information type value. Thus, each of the interim results 1212A, 1212B functions as an index into the cached content 1214.

The interim results are combined by the cache according to logical rules within the query to result in creating and storing raw results from cache 1210. For example, raw results from cache 1210 represents the logical intersection of interim results 1212A, 1212B.

Query processor 1204 receives the raw results from cache 1210 and stores them in the result cache 1206 in the form of a final result set 1208B. Query processor 1204 then delivers final result set 1208B to client 1202.

As one specific example of operation, assume that cached content 1214 contains data organized as follows:

ROWID	PRODUCT	INFO OBJ TYPE	CHUNK PATH
0	p1	intro	/a.cnk
1	p1	bene	/b.cnk
2	p1	intro	/c.cnk
3	p2	intro	/d.cnk
4	p2	bene	/e.cnk

5	p2	intro	/f.cnk
6	p1	intro	/g.cnk
7	p2	bene	/h.cnk

Each of the product values and information object values is a reference to a node object in the tree described above. The value "intro" refers to an "Introduction" information object type that is associated with information that gives an introduction to a product; the value "bene" identifies a "Features & Benefits" information type. Assume further that client 1202 issues a request for all "Features and Benefits" information objects for product p1, and the request has the form: SELECT ("p1", "bene"). Thus, the request means, "select from the cache all objects that are for product p1 and contain Features and Benefits ("bene") information." In response to this query, the cache generates interim results as follows:

FIRST INTERIM RESULTS

p1	0, 1, 2, 6
p2	3, 4, 5, 7

SECOND INTERIM RESULTS

intro	0, 2, 3, 5, 6
bene	1, 4, 7

Thus, the product attribute ("p1", "p2") and the information object type attribute ("intro", "bene") are index values, and the interim results comprise lists of all rows corresponding to all attribute values. When the interim results are received, the cache combines them according to the query such that one or more common rows of the cached content are identified. For example, the cache applies the Boolean rule "p1 AND bene" to the interim results and determines that the content with ROWID=1 satisfies the Boolean rule. Only the object with ROWID=1 is found in both of the interim results. Accordingly, result set 608B includes only information identifying row 1.

Row 1 of cached content 1214 identifies content chunk "/b.cnk." Therefore, query processor 1204 returns that content chunk to the client as the result of the query. Client 1202 may be the delivery engine 1110. Thus, in response to receiving the result of the query, comprising one or more content chunks or information objects, delivery engine 1110 can dynamically construct an electronic document that contains the information objects and is responsive to the user query.

If client 1202 issues a second query in the same form (seeking information matching "p1" and "bene"), the result cache 1206 is first searched for matching information. In the example given, the result cache has the values "p1," "bene," and "1." Thus, a search of the result cache 1206 yields a result set matching the query, so no inquiry to cached content 1214 or generation of interim result sets is needed. The row values in the client ("1" in this example) are immediately returned to the client.

In this configuration, retrieval of complex information content is carried out using highly efficient data storage. For example, cached content 1214 may be created using as few as 13 bytes per row to store a ROWID, attribute values, and chunk references. Actual information object content chunks are stored outside the cache core, as stored content chunks 1216, which are accessed only when a cache hit occurs, as indicated by information in result set 1208B. In contrast, object repository 1220 typically uses far greater storage for object references.

A cache system as described herein is optimized for operation in connection with data representing vocabulary concepts, relationships, and information objects, in this embodiment. Lookup time is minimized by using a cascading lookup scheme that first involves querying result cache 1206. Cached content 1214 and object repository 1220 are queried, and result sets are constructed, only if necessary.

In one embodiment, object repository 1220 and persistent content store 454 are configured as a set of tables in a relational database management system. FIG. 13 is a

diagram of an example schema of tables that may be used in an embodiment. Boxes represent tables, and connecting lines represent primary keys into other tables. Elements identifies as "Number" each are 4 bytes long and "String" elements are 256 characters, in an embodiment. The ID values are generated by cache servers 440.

Such a schema is highly optimized in terms of table size and memory usage. Each table has relatively few columns. A vocabulary table 1302 defines primary keys for a Relation Type table 1303, Relation Type Participant table 1304, a Relation Participant table 1306, and an Attribute table 1308. Relation types refer to relationships among information objects, e.g., one vocabulary item in Vocabulary table 1302 may have a "has_doc" relation with a particular information object, which indicates that the vocabulary item has a document associated with it. Information objects that are referenced in Info Object table 1310 may be stored in a content management system (CMS) 1312, in an LDAP directory 1314, or in a file system 1316, as indicated by an object ID value 1311 that uniquely identifies an object. Thus, the repository may reference any number of different information storage mechanisms. Ultimately, this enables a dynamically constructed electronic document based on such information objects to have a richer variety of information.

When a cache server 440 starts up, it reads these tables to build a representation or data model in memory. Further, each cache server 440 also periodically merges the changes that are applied on the in-memory data with the persistent store.

Each cache server 440 also may include an event manager and may generate events during its operations. For example, events may be generated when tree nodes are changed, or when the cache re-indexes, or upon the occurrence of any other event of interest. The event manager provides registration APIs and an event dispatch mechanism.

Each cache server 440 also may provide an administrative subsystem to support administrative functions such as stopping and starting the cache server, distribution of

physical copies of data, changing configuration parameters at runtime, log management, load balancing, mirroring, etc.

Stored content chunks 1216 and object repository 1220 may be distributed over multiple machines. For example, one hierarchical tree of information objects (such as the "Product" category tree) may be stored in one physical computing machine and other trees may be stored in other machines. Alternatively, one category may be replicated in part or whole, and may be integrated with an event manager to maintain consistency and integrity of data and relationships.

FIG. 14 is a block diagram of a distributed computing model in which hierarchies of information objects are distributed among multiple machines. A master machine 1402 stores a root node 1404, a Product proxy root node 1406, a Technology proxy root node 1408, and a Solution proxy root node 1410. Each of the proxy root nodes points to a hierarchy or tree of nodes in a particular machine. For example, Product proxy root node 1406 references a distributed root node 1418 on a first machine 1412. The distributed root node 1418 references a Product root node 1420 in which the Product hierarchy of nodes 1422 is rooted.

Similarly, Technology proxy root node 1408 and Solution proxy root node 1410 reference a distributed root node 1424 in a second machine 1414 in which both a Technology root node 1426 and Solution root node 1428 are rooted. Thus, one or more hierarchies, trees or branches may reside on a machine.

As a further performance enhancement, hierarchies of nodes may be mirrored. For example, a hierarchy of nodes may be stored as a first copy of the hierarchy and a second copy of the hierarchy. The first copy is used only for high-volume, fast read operations and the second copy is used only for slower, synchronized write operations that use node-level locking, etc., for write consistency. Periodically, the copies are synchronized to one another. For example, the second copy is replicated, the replicated second copy is designated as the read copy, and the first copy is discarded. Such synchronization may be carried out at any

convenient interval, e.g., every hour, every several hours, based on subscription to events, etc. Constraints may determine, on a node-specific level, rules for replication, as all vocabularies may have unique requirements.

Thus, a high-performance, multi-threaded, distributed caching server that handles high-volume client requests for accessing and querying concepts, relationships and info-objects is provided. Unlike prior approaches that cache static pages only and cannot account for page personalization or dynamic content, the cache disclosed herein caches components of pages that are frequently used. A delivery engine queries the cache to obtain components of pages that the delivery engine is assembling for delivery to a client. Unlike prior approaches, the delivery engine is never required to query a database or repository of content information. The de-normalized table for fetching information objects provides a composite index into the information objects, facilitating quick look-up and avoiding walking through the tree data structure.

In this embodiment, the de-normalized table implements a LRU algorithm to control the cache size of in-memory chunks. In addition, relationships and constraints in the model may affect other events that provide logic other than an LRU scheme, or in addition to it, for making the decision on what vocabularies, relationships, and information objects to cache at what layers or distributed services.

According to other features, an Event Manager notifies interested applications about what is happening in the caching system. The cache may generate events, for example, when values of tree nodes are changed, when index updates are carried out, when hierarchies are re-synchronized, etc. As a specific example, when an information object value changes, an event is published. The delivery engine 1110 subscribes to such events so that it can request a new copy of the changed information object when it next generates an electronic document that contains such information object. Examples of events that can be published include Node Added; Node Removed; Node Attribute Added; Node Attribute Removed; Relation Type

Added; Relation Type Removed; Relation Attribute Added; Relation Attribute Removed; Relation Instance Added; Relation Instance Removed.

Administrative services enable remote administration of the caching system for fine-tuning its operation and ordering its operations to stop or start. An Update Manager manages synchronization with master copies of data that are stored in a persistent repository.

The structures described herein may be implemented in one or more servers, programs, processes or other software elements, machines or other hardware elements. In one embodiment, each concept cache server 440 is implemented as a plurality of Java® classes, methods and other program elements that are compiled and linked with appropriate communication libraries, data access libraries, and similar elements to form a machine-executable system having the logical structure and functions disclosed herein.

. Vocabularies and relationships are cached with their references to other objects, as needed, facilitating speed of execution of both the logic of constructing a document and in finding the appropriate cached version of an information object. The intelligence in the data model allows numerous embodiments of more and more complex caching scenarios based on evolving architectural needs. Even a simple example configuration, as disclosed herein, solves an unmet challenge in data access today in a new and highly scalable way.

5.0 HARDWARE OVERVIEW

FIG. 10 is a block diagram that illustrates a computer system 1000 upon which an embodiment of the invention may be implemented. Computer system 1000 includes a bus 1002 or other communication mechanism for communicating information, and a processor 1004 coupled with bus 1002 for processing information. Computer system 1000 also includes a main memory 1006, such as a random access memory ("RAM") or other dynamic storage device, coupled to bus 1002 for storing information and instructions to be executed by processor 1004. Main memory 1006 also may be used for storing temporary variables or other intermediate information during execution of instructions to be executed by processor

1004. Computer system 1000 further includes a read only memory ("ROM") 1008 or other static storage device coupled to bus 1002 for storing static information and instructions for processor 1004. A storage device 1010, such as a magnetic disk or optical disk, is provided and coupled to bus 1002 for storing information and instructions.

Computer system 1000 may be coupled via bus 1002 to a display 1012, such as a cathode ray tube ("CRT"), for displaying information to a computer user. An input device 1014, including alphanumeric and other keys, is coupled to bus 1002 for communicating information and command selections to processor 1004. Another type of user input device is cursor control 1016, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 1004 and for controlling cursor movement on display 1012. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), that allows the device to specify positions in a plane.

The invention is related to the use of computer system 1000 for a vocabulary development server and information object repository. According to one embodiment of the invention, a vocabulary development server is provided by computer system 1000 in response to processor 1004 executing one or more sequences of one or more instructions contained in main memory 1006. Such instructions may be read into main memory 1006 from another computer-readable medium, such as storage device 1010. Execution of the sequences of instructions contained in main memory 1006 causes processor 1004 to perform the process steps described herein. In alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions to implement the invention. Thus, embodiments of the invention are not limited to any specific combination of hardware circuitry and software.

The term "computer-readable medium" as used herein refers to any medium that participates in providing instructions to processor 1004 for execution. Such a medium may

take many forms, including but not limited to, non-volatile media, volatile media, and transmission media. Non-volatile media includes, for example, optical or magnetic disks, such as storage device 1010. Volatile media includes dynamic memory, such as main memory 1006. Transmission media includes coaxial cables, copper wire and fiber optics, including the wires that comprise bus 1002. Transmission media can also take the form of acoustic or light waves, such as those generated during radio-wave and infra-red data communications.

Common forms of computer-readable media include, for example, a floppy disk, a flexible disk, hard disk, magnetic tape, or any other magnetic medium, a CD-ROM, any other optical medium, punch cards, paper tape, any other physical medium with patterns of holes, a RAM, a PROM, and EPROM, a FLASH-EPROM, any other memory chip or cartridge, a carrier wave as described hereinafter, or any other medium from which a computer can read.

Various forms of computer readable media may be involved in carrying one or more sequences of one or more instructions to processor 1004 for execution. For example, the instructions may initially be carried on a magnetic disk of a remote computer. The remote computer can load the instructions into its dynamic memory and send the instructions over a telephone line using a modem. A modem local to computer system 1000 can receive the data on the telephone line and use an infra-red transmitter to convert the data to an infra-red signal. An infra-red detector can receive the data carried in the infra-red signal and appropriate circuitry can place the data on bus 1002. Bus 1002 carries the data to main memory 1006, from which processor 1004 retrieves and executes the instructions. The instructions received by main memory 1006 may optionally be stored on storage device 1010 either before or after execution by processor 1004.

Computer system 1000 also includes a communication interface 1018 coupled to bus 1002. Communication interface 1018 provides a two-way data communication coupling to a network link 1020 that is connected to a local network 1022. For example, communication

interface 1018 may be an integrated services digital network ("ISDN") card or a modem to provide a data communication connection to a corresponding type of telephone line. As another example, communication interface 1018 may be a local area network ("LAN") card to provide a data communication connection to a compatible LAN. Wireless links may also be implemented. In any such implementation, communication interface 1018 sends and receives electrical, electromagnetic or optical signals that carry digital data streams representing various types of information.

Network link 1020 typically provides data communication through one or more networks to other data devices. For example, network link 1020 may provide a connection through local network 1022 to a host computer 1024 or to data equipment operated by an Internet Service Provider ("ISP") 1026. ISP 1026 in turn provides data communication services through the world wide packet data communication network now commonly referred to as the "Internet" 1028. Local network 1022 and Internet 1028 both use electrical, electromagnetic or optical signals that carry digital data streams. The signals through the various networks and the signals on network link 1020 and through communication interface 1018, which carry the digital data to and from computer system 1000, are exemplary forms of carrier waves transporting the information.

Computer system 1000 can send messages and receive data, including program code, through the network(s), network link 1020 and communication interface 1018. In the Internet example, a server 1030 might transmit a requested code for an application program through Internet 1028, ISP 1026, local network 1022 and communication interface 1018. In accordance with the invention, one such downloaded application provides for an information object repository API as described herein.

The received code may be executed by processor 1004 as it is received, and/or stored in storage device 1010, or other non-volatile storage for later execution. In this manner, computer system 1000 may obtain application code in the form of a carrier wave.

6.0 EXTENSIONS AND ALTERNATIVES

In the foregoing specification, the invention has been described with reference to specific embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.
